Санкт-Петербургский государственный университет

Кафедра вычислительных методов механики деформируемого тела

Максимов Анатолий Александрович

Выпускная квалификационная работа бакалавра

Адаптация алгоритмов пакета Geant4 на вычислительную архитектуру HybriLIT

Направление 010300

Фундаментальная информатика и информационные технологии

Научный руководитель,

старший преподаватель

Кулабухова Н.В.

Санкт-Петербург

2016

Содержание

Введение

Уже не первый десяток лет практически все научные открытия делаются большими командами учёных. В наши дни сколько-нибудь известные имена принадлежат в основном популяризаторам науки вроде Лоуренса Краусса и Стивена Хокинга. Так, в частности, в публикации с описанием первых экспериментальных данных о бозоне Хиггса большую часть статьи занимает перечисление учёных, причастных к знаменательному открытию.

Современные исследования направлены на ещё более глубинное познание устройства мира, и если раньше для необходимых экспериментов требовались скромные установки вроде микроскопа или барокамер, то сейчас наука проникла настолько глубоко, что для получения требуемых результатов нужны гораздо более сложные и, соответственно, дорогие установки – коллайдеры, циклотроны и др. И такие установки, очевидно, потребляют колоссальное количество энергии. Так, например, Большой адронный коллайдер потребляет приблизительно 180 МВт. Каждый запуск обходится очень дорого, поэтому важно сократить количество запусков до необходимого минимума, просчитав все параметры заранее.

Как раз для этих задач необходимо использовать инструменты для моделирования, чтобы правильно подобрать параметры запуска. Для точной калибровки оборудования необходимо произвести тысячи и даже миллионы тестовых прогонов до реального запуска установки, и для этого нужна мощная вычислительная техника вроде популярных сейчас кластеров и grid-систем с установленным на ней программным обеспечением для моделирования экспериментов. Одним из таких инструментов является пакет Geant4, разработанный специалистами исследовательской организации ЦЕРН (CERN – Conseil Européen pour la Recherche Nucléaire, с фр. Европейский совет по ядерным исследованиям).

Geant4 – инструментарий для моделирования прохождения элементарных частиц через материю методами Монте-Карло. Он обладает широкой функциональностью, включающую трекинг, построение геометрии, физических моделей и т.д. Физические процессы затрагивают внушительное количество областей, в том числе электромагнитные, адронные и оптические процессы, огромное множество долгоживущих частиц, материалов и элементов в некоторых случаях с энергией от 250 эВ и иногда до ТэВ. Geant4 был построен, чтобы работать с физическими моделями, обрабатывать сложную геометрию и давать возможность адаптировать всё это максимально просто для оптимального использования в различных приложениях [6].

Данный инструментарий сочетает в себе многофункциональность и простоту в написании приложений благодаря объектно-ориентированному подходу, реализованному на языке программирования C++, что позволяет использовать его даже специалистам, которые не обладают глубокими познаниями в информационных технологиях и не владеют высокими профессиональными навыками программирования, в самых различных областях науки и промышленности для моделирования объектов и процессов разного рода – от ускорительной техники вроде Большого адронного коллайдера до позитронной томографии и адронной терапии.

Первая версия Geant4 была выпущена ещё в 1995 году, с тех пор инструментарий стал основной программой моделирования в экспериментах на LHC (кроме ALICE), а также получил широкое распространение среди множества отраслей. Эксперименты стали гораздо сложнее, и для повышения эффективности пакета в десятой версии была добавлена поддержка многопоточности – что означило рывок в сторону высокопроизводительных распределённых вычислений. Некоторые эксперименты в Geant4 могут занимать часы, дни и даже недели вычислений на очень энергозатратном оборудовании, и остро встал вопрос экономии как вычислительного времени, так и, следовательно, средств.

Итак, для решения данной проблемы видится три пути:

- покупка более производительного оборудования;
- оптимизация кода на конкретную архитектуру;
- поиск оптимальных конфигураций сборки пакета.

Первый вариант требует больших финансовых затрат, второй – временных. Очевидно, способ одновременно экономичный и быстрый – последний.

Данная проблема будет решаться относительно гетерогенного кластера HybriLIT, расположенного в г. Дубна в Лаборатории Информационных Технологий Объединённого Института Ядерных Исследований (далее ЛИТ ОИЯИ). Архитектура гетерогенного кластера является одной из самых популярных среди суперкомпьютеров. Так, подавляющее большинство суперкомпьютеров из ТОП500 имеют кластерную архитектуру.

Постановка задачи

Целью данной работы является повышение эффективности работы пакета Geant4 на гетерогенном кластере HybriLIT. Для достижения поставленной цели предполагается протестировать работу инструментария на сборках под разными компиляторами и на сопроцессоре Intel Xeon Phi.

Решение данной задачи требует реализации следующих этапов:

1) компиляция пакета Geant4 под разными компиляторами с разными флагами оптимизации;
2) компиляция примеров из директории с исходным кодом Geant4 на соответствующих сборках Geant4;
3) запуск примеров на кластере:
   а. На различном количестве событий;
   б. На различном количестве потоков;
   в. На CPU и сопроцессоре Intel Xeon Phi;
4) анализ полученных результатов:
   а. Вычисление ускорения (speed-up) на каждом из экспериментов;
   б. Вычисление эффективности;
   в. Сравнение полученных на разных сборках данных;
   г. Поиск оптимальных конфигураций.

Сформулированная задача будет решаться в рамках совместного проекта СПбГУ и ЛИТ ОИЯИ, в которой и расположен гетерогенный кластер HybriLIT, и является важной частью научного исследования, проводимого в рамках данного проекта.

Глава 1. Описание предметной области

Предметная область представляет собой программный пакет Geant4, описание его структуры, принципа работы, а также описание структуры приложений для него.

1.1. Описание пакета Geant4

1.1.1. Описание физических процессов

Основное направление Geant4 – моделирование задач физики высоких энергий, или физики элементарных частиц, т.е. рассмотрение природы на очень малых расстояниях с целью исследования элементарных составляющих веществ и их взаимодействий.

Чем глубже проникновение в материю, тем больше требуется энергии для сталкивающихся частиц, так как по принципу неопределённости существует связь между энергией и расстоянием, при которых происходят процессы в микромире. Принцип неопределённости – это фундаментальный закон в квантовой теории, который гласит, что так называемые дополнительные физические величины вроде энергии и времени, характеризующие физическую систему, не могут одновременно принимать точные значения ($\Delta p \Delta x \sim \hbar$, где $\hbar$ – постоянная Планка). Это отражает двойственную, корпускулярно-волновую природу элементарных частиц (волновые свойства материи были открыты французом-физиком Луи де Бройлем в 1924 г.).

Всегда увеличение энергии сопровождалось раскрытием совершенно новых физических явлений, и это также отражено в пакете Geant4 – в зависимости от расстояния и энергии частиц происходят различные взаимодействия:

- молекулярная физика – 10^{-7} – 10^{-9} м, энергии ~ 1 эВ – 1 кэВ;

- атомная физика, антиматерия – 10^{-13} м, энергии ~10 МэВ;
- ядерная физика – 10^{-15} м, энергии 100 МэВ – 1 ГэВ;
- физика высоких энергий, адронная физика – 10^{-15} – 10^{-17} м, энергии ~ 10 – 100 ГэВ;
- физика «сверхвысоких» энергий, процессы на сверхмалых расстояниях, меньших 10^{-17} м, энергии больше 100 ГэВ.

На Рис. 1 наглядно показаны соотношения расстояний, времени жизни и энергий для ядер и частиц [13].

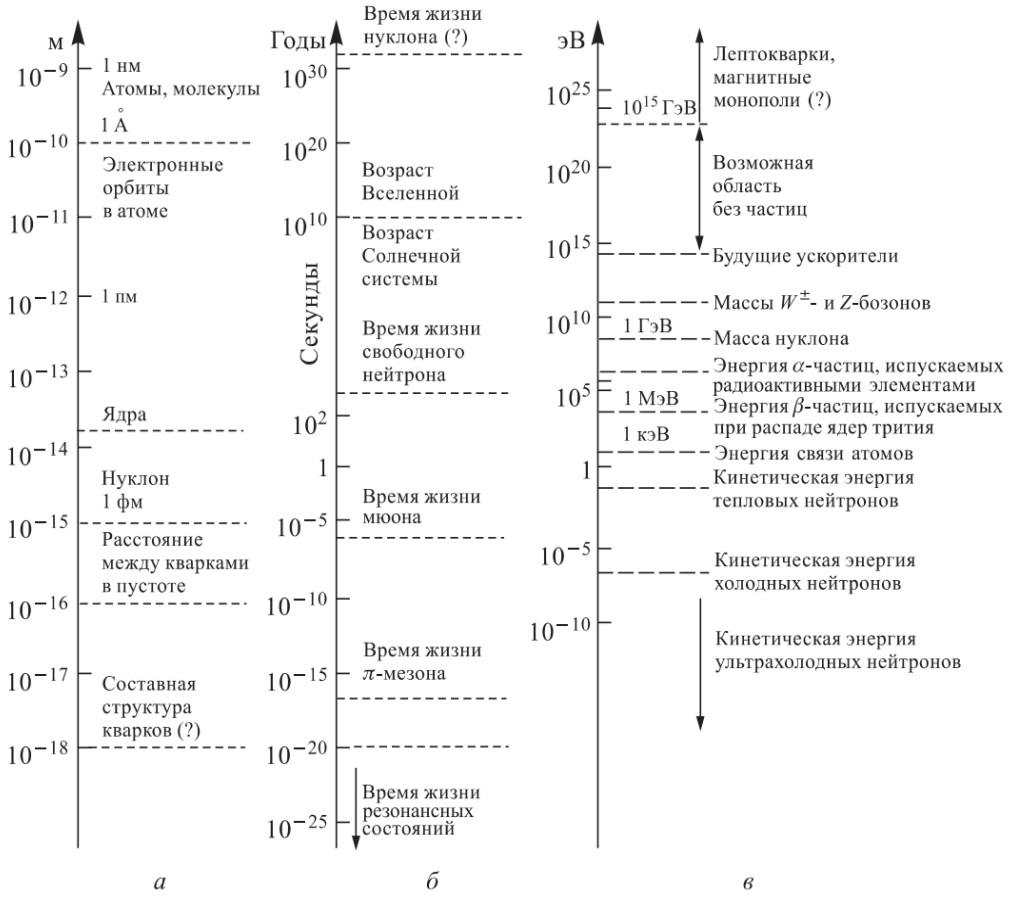

Рис. 1. Масштабы расстояний (а), времени (б) и энергий (в) в физике ядер и частиц

Молекулярная физика

Взаимодействия молекулярных пучков и воздействие на них электромагнитными полями играет важную роль в современных физических исследованиях. Так, например, один из способов разделения изотопов состоит

в воздействии электрическим и магнитным полями на пучок молекул, состоящий из смеси ионизированных атомов различных изотопов. Различные ускорители, с помощью которых исследуется атомное ядро, также используют пучки ионов, разгоняемые до больших скоростей в электрическом и магнитном полях [15].

Одним из первых опытов, относящихся к молекулярной физике, является опыт Штерна 1920 года, задачей которого было измерение движения скорости молекул газа. Устройство Штерна представляло собой цилиндр, в центре которого располагалась посеребренная платиновая нить, через которую пропускался электрический ток, и часть молекул серебра, которые пропускала небольшая щель, летела прямолинейно и равномерно к охлаждённой внутренней поверхности цилиндра, оставляя там след. Таким образом скорость молекул можно было измерить по формуле $v = \frac{ul}{s}$, где s – длина дуги между точками следа в состоянии покоя цилиндра и при движении; l – расстояние между щелью и следом в состоянии покоя цилиндра; u – скорость движения точек цилиндра [29].

Атомная физика

Атомная физика, по большому счёту, зародилась ещё в Древней Греции в трудах Демокрита и других философов, как идея о существовании мельчайших неделимых частиц. Однако одним из самых важных событий стало описание модели атома Дж. Дж. Томсоном в 1903 году на основе открытий Марии Склодовской-Кюри и Пьера Кюри [20]. Однако самыми важными методами изучения строения атома являются исследования эффектов, полученных направлением на атом электромагнитных волн или пучков частиц, а также при столкновении атомов на больших скоростях [5]. Так, в 1911 году Эрнест Резерфорд бомбардировал альфа-частицами золото, доказывая делимость атома, а в современности столкновения атомов и молекул применяют для ионизации [17].

Однако благодаря экспериментам со столкновением атомов были открыты также и ранее отсутствующие элементы в периодической таблице Менделеева. Так, например, в 1968 году в наукограде Дубна в Лаборатории Ядерных Реакций ОИЯИ командой учёных во главе с Флёровым Георгием Николаевичем при бомбардировке америция (^{243}Am) ионами неона (^{22}Ne) была обнаружена случайная активность распада со временем полураспада 1,8±0,6 секунд, так был впервые получен 105 элемент, впоследствии названный дубнием [3].

Ядерная физика

Большое влияние на ядерную физику оказал, как и в случае с атомной физикой, Резерфорд. В 1919 году он при столкновении α-частиц с азотом обнаружил, что азот распался на две частицы – кислород и ещё одна, предположительно водород, однако впоследствии оказалось, что это был протон [26]. Это, кстати, была первая искусственная трансмутация элементов. В дальнейшем Резерфордом были проведены ядерные реакции в 17 элементах, включая бор, фтор, фосфор и др.

В современной же ядерной физике доминируют следующие направления [16]:

- генезис новых сверхтяжёлых ядер;
- анализ свойств ядерной материи при низких и высоких температурах, низкой и высокой плотности – в экстремальных условиях;
- изучение формы и свойств ядер в супердеформированных состояниях и в состояниях с экстремально большими спинами;
- исследование новых типов радиоактивного распада;
- нуклон и его кварк-глюонная структура, изменение его свойств в ядерной материи;
- и др.

Адронная физика

Данная область физики элементарных частиц проникает ещё глубже в природу материи и изучает строение и свойства ещё более мелких частиц, нежели протоны, нейтроны, электроны, нейтрино и фотоны. Новые частицы были обнаружены после экспериментов на очень больших ускорителях в виде сотен нестабильных состояний частиц – адронов, сильно взаимодействующих частиц. Они состоят из точечноподобных фермионов – лептонов, имеющих целый электрический заряд, и кварков, несущих дробный заряд (например, $-\frac{1}{3}$e). Так, протоны и нейтроны состоят из трёх кварков [24].

Так как адронная физика – область довольно молодая, многие её элементы являются объектом самых тщательных исследований. Самый известный ускоритель, построенный для решения задач адронной физики – Большой адронный коллайдер. С одной из важнейших задач – поиском бозона Хиггса – БАК уже помог справиться учёным. Далее учёные из ЦЕРНа планируют увидеть неизвестные, ранее не наблюдавшиеся явления, а также поиск суперсимметрии, дополнительных размерностей пространства-времени, а также других, пока не предсказываемых теорией явлений [19].

Физика сверхвысоких энергий

О физике сверхвысоких энергий известно ещё меньше, так как детектировать подобные взаимодействия довольно сложно, а энергию такого порядка учёные получили сравнительно недавно. Основная часть исследований посвящена нейтрино – частицам, участвующим только в слабом и, возможно, в гравитационном взаимодействии. Первые догадки об их существовании были сформулированы в 1930 году Вольфгангом Паули, однако сейчас учёные могут регистрировать нейтрино, правда, с помощью внушительных дорогих установок вроде нейтринного телескопа IceCube, представляющего собой кубический километр ледникового льда, расположенного на Южном полюсе [10].

И хотя ежесекундно на Земле через квадратный сантиметр проходит около шестидесяти миллиардов нейтрино, зарегистрировать их очень тяжело, т.к. они оказывают минимальное влияние на вещество. Так, в 2013 году нейтринный телескоп IceCube смог зарегистрировать всего 28 нейтрино, обладающих сверхвысокой энергией: 26 с энергией от 50 ТэВ до 1 ПэВ и 2 от 1 ПэВ до 2 ПэВ. При этом источников космических лучей довольно много: начиная солнцем и заканчивая сверхновыми [8].

В будущем исследование нейтрино повлечёт развитие нейтринной астрономии [28], поможет изучить внутренний состав Земли (на основе изучения нейтрино, образованных в результате распада радиоактивных элементов в центре Земли) [25], а также, возможно, послужит толчком в развитии средств связи [21].

1.1.2. Описание алгоритмов

Geant4 объединяет в себе огромное множество алгоритмов, предназначенных каждый для своей конкретной области, которых в инструментарии множество. Для разных частиц и соответствующих энергий требуются разные методы моделирования. В их основе лежит один общий метод – метод Монте-Карло. Все методы в Geant4 представляют собой различные композиции и отклонения методов Монте-Карло. Далее будет кратко описан общий принцип, для более подробного описания авторы пакета рекомендуют обратиться к публикациям Бутчера и Месселя, Месселя и Кроуфорда или Форда и Нельсона.

Предположим, мы хотим получить x из интервала $[x_1, x_2]$ на распределении $f(x)$, и нормализованная плотность вероятности может быть записана следующим образом:

$$f(x) = \sum_{i=1}^{n} N_i f_i(x) g_i(x)$$

где $N_i > 0, f_i(x)$ – нормализованные функции плотности вероятности на $[x_1, x_2]$, и $0 < g_i(x) \leq 1$.

Согласно этого метода, x может быть получено следующим путём:

1. Выбирается случайное целое число $i \in \{1,2, \dots, n\}$ с вероятностью, пропорциональной N_i;
2. Выбирается значение x_0 на распределении $f_i(x)$;
3. Вычисляется $g_i(x_0)$ и принимается $x = x_0$ с вероятностью $g_i(x_0)$;
4. Если x_0 отклоняется, вернуться к шагу 1.

Можно показать, что эта схема верна, и среднее количество попыток получить значение равно $\sum_i N_i$.

На практике, хороший метод получения чисел из распределения $f(x)$ обладает следующими качествами:

- для любого подраспределения $f_i(x)$ можно легко провести сэмплирование;
- отклонение функций $g_i(x)$ может быть легко и быстро оценено;
- среднее количество попыток небольшое.

Таким образом, различные возможные декомпозиции распределения $f(x)$ не эквивалентны с практической точки зрения (например, они могут быть разными с точки зрения скорости вычисления), и возможно будет полезно оптимизировать декомпозицию.

Замечание, имеющее практическую важность: если наше распределение не нормализовано

$$\int_{x_2}^{x_1} f(x)dx = C > 0,$$

метод может быть использован тем же способом, среднее количество попыток в таком случае будет равняться значению $\sum_i N_i/C$ [7].

1.1.3. Описание архитектуры

Geant4 разработан при применении техник продвинутой софтверной разработки, основанных на Объектно-ориентированной методологии Booch/UML, и следовании эволюции ESA Software Engineering Standards в процессе разработки. Использован «спиральный», или итеративный, подход [12].

Geant4 писался под софтверные нужды современных экспериментов. Как и любая обычная система программного обеспечения, он содержит следующие компоненты: генератор событий, симуляция детектора, реконструкция и анализ. И все эти компоненты могут быть использованы как отдельно, так и в различных комбинациях. Данный инструментарий был построен в качестве базиса для симуляционной компоненты. Таким образом, он должен был:

- обладать компонентами со строго определёнными интерфейсами;
- обеспечивать взаимодействие своих частей с компонентами.

Другие важные требования к дизайну, которые ставились при разработке, это модульность и гибкость, а также ясная, «прозрачная» реализация физики, открытая для проверки пользователем. Это должно позволять пользователю понимать, настраивать и расширять инструментарий во всех его аспектах [1].

Ключевые элементы симуляции прохождения частиц через материю являются:

- геометрия и материалы;
- взаимодействие частиц в материи;
- управление трекингом;
- оцифровка и обработка срабатываний;
- управление событиями и треком;
- визуализация и фреймворк для визуализации;

- пользовательский интерфейс.

Эти элементы естественно ложатся на создание классов категорий со связными интерфейсами, и для каждой категории своя рабочая группа с чётко определённой «ответственностью». Это также ложится на концепцию «тулкита» (с англ. toolkit – инструментарий), который подразумевает, что пользователь может самостоятельно собирать свою программу во время компиляции из необходимых компонент, присутствующих в Geant4 или предоставленных самостоятельно. Схема отношений классов проиллюстрирована на Рис. 2.

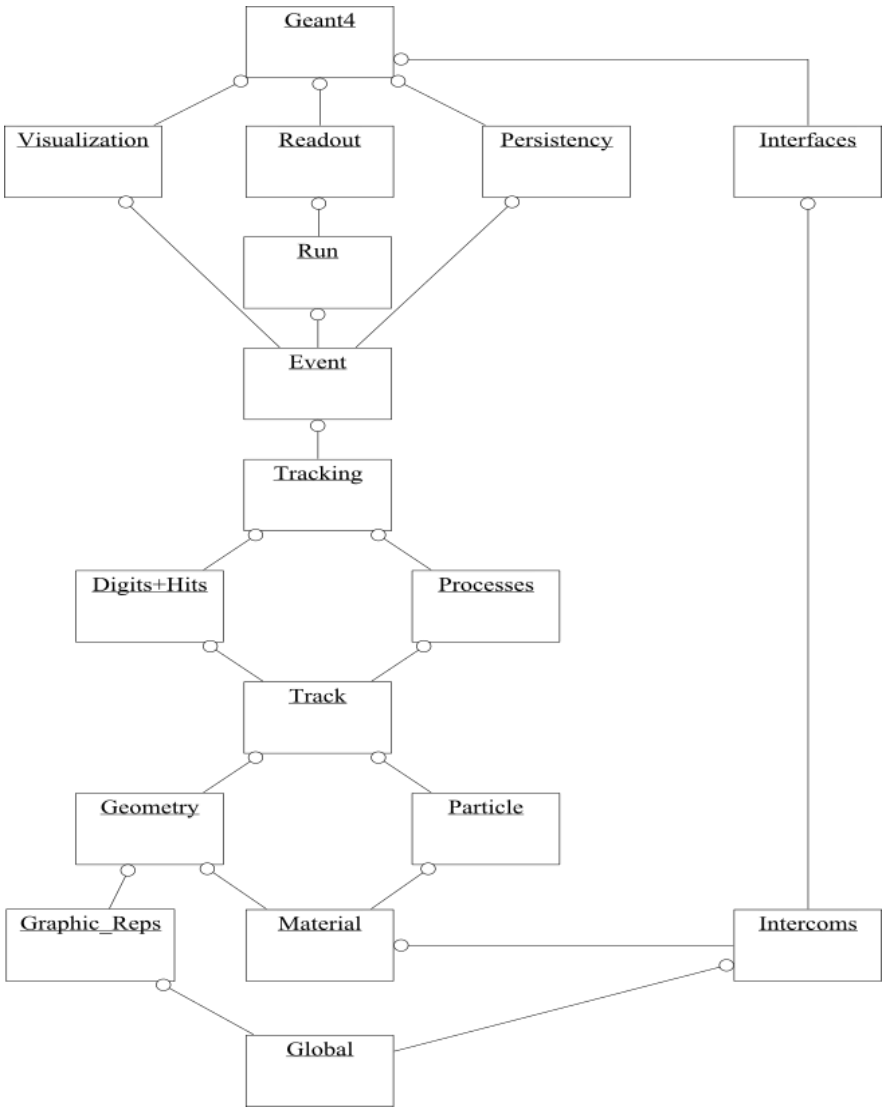

Рис. 2. Диаграмма категорий верхнего уровня пакета Geant4. Класс на конце кружка использует присоединённый класс.

Дизайн эволюционировал в течение разработки, и сейчас он представляет собой 17 больших категорий, определённых анализом пользовательских требований [11]. Взглянув на рисунок 2, можно увидеть, что это однонаправленный орграф зависимостей без циклов, как и должно быть. Видно, что категории в нижней части диаграммы используются во всех более высоких категориях и по сути обеспечивают фундамент Geant4 [1].

Таким фундаментом являются следующие категории:

- *global*, включающая в себя систему единиц (при этом особенность дизайна заключается в независимости от внутреннего представления величин), константы, числа и обработку случайных величин;
- *materials*, описывающая материалы*;*
- *particles*, описывающая частицы*;*
- *graphical representations*, описывающая графическое представление*;*
- *geometry*, включающая в себя объёмы для описания детектора и навигации в геометрической модели;
- и *intercoms*, обеспечивающая одновременно общение пользователя с Geant4 через пользовательский интерфейс, а также взаимодействие между модулями, которые также не должны зависеть друг от друга; в дополнение предоставляющая репозиторий абстрактных интерфейсов для плагинов.

Уровнем выше находятся категории, необходимые для описания трекинга частиц и физических процессов, которым они подвержены. Категория *track* включает классы для треков и шагов, используемые категорией *processes*, которая содержит реализацию моделей физических взаимодействий. Вдобавок, один такой процесс, *transportation*, обеспечивает транспортировку частиц в геометрической модели и, опционально, допускает срабатывание параметризации.

Все эти процессы могут быть вызваны категорией *tracking*, которая управляет их влиянием на состояние трека и в значительном объёме обеспечивает информацией для попаданий и оцифровки.

Над ними категория *event* управляет событиями в терминах их треков, а также *run*, управляющая коллекциями событий, которые составляют общий пучок и реализацию детектора. Категория *readout* позволяет решать проблему «нагромождений» пучков.

В конце концов, все возможности, которые задействуют всё выше описанное и взаимодействуют с внешними дополнениями, обеспечены работой категорий *visualisation*, *persistency* и [user] *interface*.

1.1.4. Основные термины Geant4

Run (сеанс) – самый крупный элемент моделирования, представляющий собой последовательность событий. Во время него условия эксперимента (набор физических процессов и геометрическая модель) остаются неизменными. Представлен классом G4Run, управляется экземпляром класса G4RunManager (или его аналогами: G4MTRunManager в многопоточной версии, tbbMasterRunManager при использовании библиотеки Thread Building Blocks и др.; далее при упоминании G4RunManager будут иметься в виду в том числе любые из аналогов).

Event (событие) – единичное независимое измерение физического явления детектором. Описывающий событие класс – G4Event – содержит все входные и выходные данные смоделированного события. Экземпляр G4RunManager создаёт объект класса G4Event, который затем передаётся объекту класса G4EventManager для осуществления управления событием. Структура события следующая:

- первичное состояние (вершина и частица);
- траектории;
- коллекция срабатываний;

- коллекция оцифрованных сигналов.

Step (шаг) – минимальное продвижение частицы сквозь вещество в условиях протекания различных физических процессов. Описывается классом G4Step.

Track (трек) – траектория движения частицы. В экземпляре класса G4Track хранится информация о полном продвижении частицы в веществе на момент обращения, а также описание частицы, текущего физического процесса и геометрического объёма.

Hit (срабатывание) – описание одного взаимодействия частицы с веществом в области детектирования – координаты, энергия, импульс в момент взаимодействия, а также время взаимодействия. Моделирование оцифрованного сигнала происходит на основе именно этой информации.

Digi (оцифрованный сигнал) – результат обработки срабатываний в виде «канал-сигнал», полный аналог величин, измеряемых в реальном эксперименте. Может быть получен в результате нескольких срабатываний [22].

1.1.5. Описание приложения на Geant4

Любое приложение на Geant4 обязательно должно содержать описание:

- распределение вещества в детекторе и поле;
- генератор начальной точки трека;
- список физических процессов, участвующих в эксперименте.

Дополнительно могут быть описаны:

- чувствительные элементы и способы моделирования отклика;
- методы визуализации;
- графический интерфейс;
- пользовательские расширения;
- и т.д.

Примитивная блок-схема моделирования любого эксперимента продемонстрирована на Рис. 3 [23].

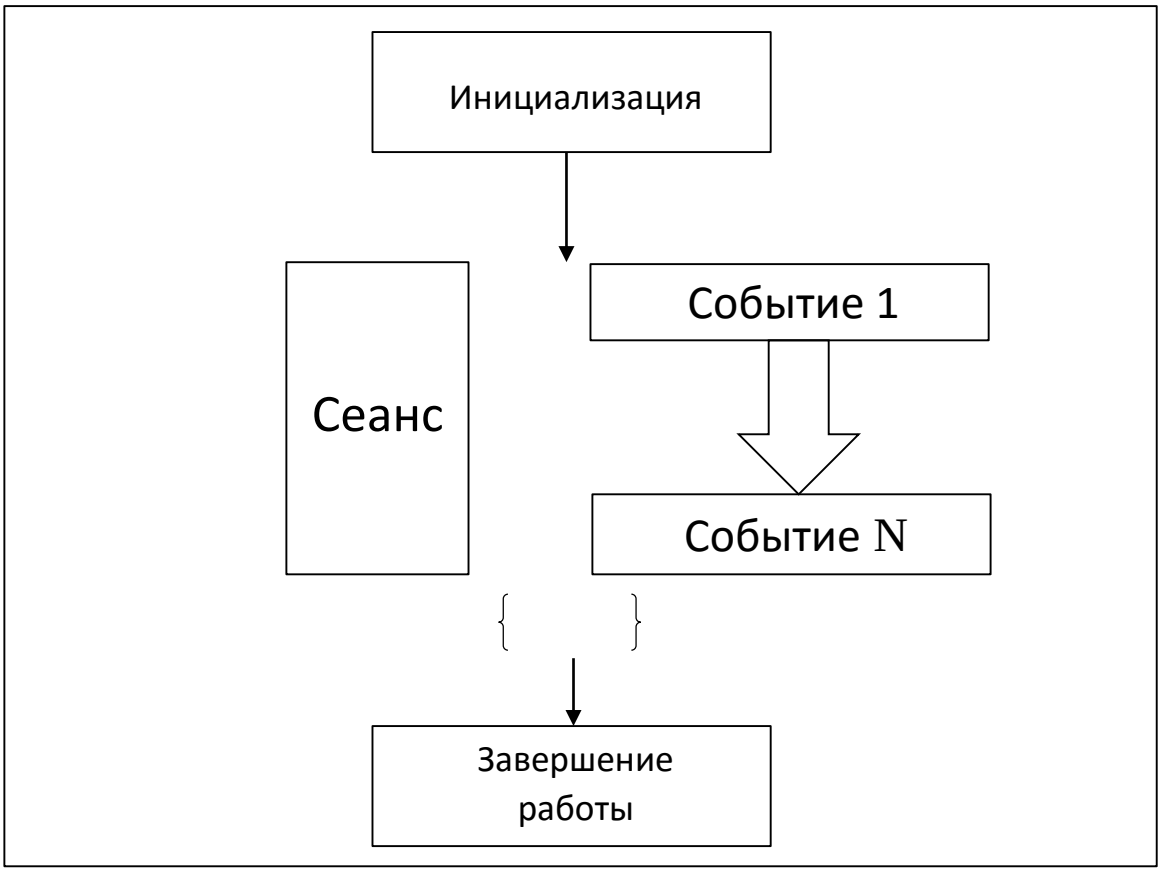

Рис. 3. Блок-схема работы любой программы на Geant4.

Пример примитивной программы на Geant4:

```
#include "G4RunManager.hh"
#include "ExampleDetectorConstruction.hh"
#include "ExamplePhysicsList.hh"
#include "ExamplePrimaryGeneratorAction.hh"
int main()
{
 G4RunManager* runManager = new G4RunManager;
 runManager->SetUserInitialization(new ExampleDetectorConstruction);
 runManager->SetUserInitialization(new ExamplePhysicsList);
 runManager->SetUserAction(new ExamplePrimaryGeneratorAction);
 runManager->initialize();
 int numberOfEvent = 10;
 runManager->BeamOn(numberOfEvent);
```

```
 delete runManager;
 return 0;
}
```

1.2. Описание гетерогенного кластера HybriLIT

Кластер HybriLIT введён в эксплуатацию в 2014 год в качестве части Многофункционального информационно-вычислительного комплекса в ЛИТ ОИЯИ. Для разработки параллельных приложений была выбрана гетерогенная структура вычислительных узлов. Такое выбор архитектуры был сделан для ускорения решения различных ресурсоёмких математических задач благодаря наличия возможности использовать потенциал многоядерных компонент и ускорителей: сопроцессоров Intel Xeon Phi и графических процессоров NVIDIA.

1.2.1. Описание аппаратного обеспечения

HybriLIT состоит из девяти вычислительных узлов:

- каждый из которых оснащён двумя ЦП Intel Xeon;
- на трёх блэйдах установлены сопроцессоры Intel Xeon Phi: один с Intel Xeon Phi 5110P, один с Intel Xeon Phi 7120P и два с двумя Inter Xeon Phi 7120P;
- на семи узлах стоят видеокарты NVIDIA Tesla: один узел с NVIDIA Tesla K20X, четыре узла с тремя NVIDIA Tesla K40 и два узла с двумя NVIDIA Tesla K80.

Общий объём оперативной памяти составляет 896 Гб. Объём дисковой памяти составляет 57,6 Тб.

Более подробная схема изображена в приложении 1.

1.2.2. Описание программного обеспечения

Кластер работает под управлением операционной системы Scientific Linux 6.7 на файловой системе NFS4 [18].

В качестве планировщика задач установлена утилита SLURM Workload Manager (Simple Linux Utility for Resource Management – с англ. простая утилита для управления ресурсами Linux). Для отправки задачи в очередь необходимо написать скрипт и запустить его командой *sbatch <название скрипта>*.

Простейшая структура скрипта выглядит следующим образом (полный набор команд можно посмотреть на сайте планировщика задач SLURM):

```
#!/bin/sh
#SBATCH -p <тип вычислительного узла (partition)
#SBATCH -w <название вычислительного узла>
#SBATCH -c <количество ядер процессора>
#SBATCH -N <количество вычислительных узлов>
./<название исполняемого файла>
```

Для динамичного изменения пользовательского окружения установлен пакет Environment Modules. Этот пакет позволяет быстро настраивать переменные окружения, подключая нужные модули. Так, например, чтобы подготовить кластер к работе с компилятором GCC версии 4.8, достаточно выполнить команду *module add gcc-4.8*. Так, на выбор пользователя предоставлено множество модулей с компиляторами разных версий GNU, PGI, Intel, CUDA и др. Помимо компиляторов есть также различные утилиты вроде ROOT, cmake и мн. др. Для написания параллельных программ доступны технологии MPI, CUDA, OpenMP и OpenCL. Чтобы увидеть список доступных модулей, можно воспользоваться командой *module avail* [18].

Для доступа к репозиторию ЦЕРНа установлен пакет CVMFS, который даёт возможность расширить список доступного программного обеспечения

кластера. Используемые пакеты подгружаются на вычислительные узлы по мере использования [14].

1.2.3. Описание информационной среды

Для эффективной работы пользователей и доступа к необходимой и полезной информации о HybriLIT была разработана информационно-вычислительная среда [15], в состав которой входят следующие сервисы:

- *веб-сайт HybriLIT* [18]– ресурс, содержащий подробную информацию об аппаратном и программном обеспечении, руководство пользователя, а также инструкцию для регистрации новых пользователей на кластере;
- *система Indico* (http://indico-hybrilit.jinr.ru) для организации конференций, семинаров и встреч группой HibriLIT в рамках института;
- *HybriLIT User Support* (https://pm.jinr.ru/projects/hybrilit-user-support) – проект в системе Project Management Service, позволяющий осуществлять техническую поддержку пользователей, а также размещать информацию о грядущих мероприятиях, связанных с кластером, а также новости и полезные материалы;
- *GitLab* (http://gitlab-hybrilit.jinr.ru)– сервис для совместной параллельной разработки приложений, представляющий себе систему контроля версий с широким функционалом.
- *Система мониторинга* (*http://stat-hlit.jinr.ru*) – сервис, который позволяет системным администраторам следить за работой на кластере и его загрузкой, а простым пользователям предоставляет статистику работы кластера и функции для просмотра информации о запущенных задачах (так, например, можно удалённо смотреть за прогрессом выполнения задач с мобильных устройств).

Глава 2. Тестирование пакета Geant4 на примерах

2.1. Подготовка окружения

Для тестирования выбрана версия пакета Geant4.10.1.p02, как последняя стабильная на момент начала исследований. Для начала исходный код пакета, доступный на кластере по адресу */cvmfs/geant4.cern.ch/geant4/10.1.p02/share*, был скопирован в пользовательскую директорию для дальнейших компиляций под разными компиляторами. В моём случае это была директория *~/Geant/geant4-src*. Далее под каждую сборку были созданы каталоги, соответствующие названию компилятора и флагу оптимизации, если таковой имел место. Таким образом, сборке под компилятором GNU с флагом оптимизации -O3 соответствует директория geant4-gcc-opt3.

Сборка пакета [21] состоит из нескольких этапов:

1. Указание переменных окружения для компилятора.

```
$ export CC=<путь к компилятору C>
$ export CXX=<путь к компилятору C++>
```

2. Создание папки и предварительная сборка пакета.

```
$ mkdir build
$ cd build
$ cmake -DGEANT_INSTALL_DATA=ON /
-DGEANT_INSTALL_PREFIX=~/geant4-gcc-opt3 /
-DGEANT_MULTITHREADED=ON -O3 ../build
```

3. Установка пакета.

```
$ make
$ make install
```

Можно также указать опцию -jN, чтобы распараллелить компиляцию на N (вместо N поставить любое натуральное число) потоков.

После этого этапа директорию build можно удалить.

4. Добавление Geant4 в переменные окружения.

```
$ source ~/Geant/geant4-gcc-opt3/bin/geant4.sh
```

После выполнения этапов можно приступить к работе с приложениями.

Для приложений была создана отдельная папка в домашней директории под названием ex-s (от англ. examples – примеры). Внутри неё отдельные папки для различных примеров. Например, TestEm12. Внутри папок с названиями примеров – папки с исходным кодом и сборками под различными компиляторами.

Сборка приложения состоит из одного шага: создание папки и сборка билда [4]:

```
$ mkdir B12-gcc-opt3-build
$ cd B12-gcc-opt-build
$ cmake ../src
$ make
```

После чего приложение готово к запуску.

Для тестирования были выбраны компиляторы GCC версии 4.9.1 и ICC из Intel Cluster Studio 2013, как самые новые из доступных на кластере на момент начала исследования.

2.2. Использованные формулы

При анализе результатов были использованы следующие формулы:

- ускорение: $S_N = \frac{T_1}{T_N}$, где $T_N -$ время работы на N потоках;
- эффективность: $E_N = \frac{S_N}{N}$.

2.3. Тестирование простого примера

В качестве начального примера был выбран пример TestEm12 из директории /examples/extended/electromagnetic/ в папке с исходным кодом

Geant4. Выбор был сделан именно таким образом, чтобы иметь в дальнейшем возможность сопоставить полученные результаты с уже имеющимся исследованием [9].

2.3.1. Описание примера

Задача: построение профиля глубинной дозы в сферической геометрии.

Описание геометрии

Геометрия представляет собой сферу из однородного материала. По желанию сферу можно разделить на слои.

Геометрию определяют три параметра:

- материал сферы;
- радиус сферы;
- количество слоёв.

В дополнение ко всему можно описать поперечное магнитное поле.

Лист физики

Включает в себя следующие модули:

1. Транспортировка.
2. Электромагнитная физика.
3. Распад.
4. StepMax (ограничение шага).

Состав события

Начальной точкой служит частица, случайным образом выпущенная из центра сферы. Тип частицы и её энергия описаны в классе *PrimaryGeneratorAction*, а также могут быть изменены с помощью команд в пользовательском интерфейсе или в макросе запуска.

По желанию пользователя можно выключить случайность в выборе направления полёта частицы.

Трекинг

Данный пример вычисляет общую энергию, потерянную частицей вдоль траектории полёта – так называемый продольный профиль энергии, или распределение глубинной дозы. Потерянная энергия распределяется вдоль шага случайным образом.

Максимальный размер шага заряженной частицы вычисляется автоматически (подробнее в описании класса RunAction), чтобы поддерживать точность вычислений.

Гистограммы

В данном примере предопределены 12 различных одномерных гистограмм, но из них по умолчанию выбраны лишь три:

- профиль энергии dE/dr (в МэВ/мм на событие) (1);
- общая длина первичного трека (3);
- нормализованный профиль энергии $d\left(\frac{E}{E_0}\right)/d\left(\frac{r}{r_0}\right)$, где r_0 – радиус действия первичной частицы с энергией E_0 (8).

Полученные замеры в оцифрованном виде записываются в файл формата root. С помощью утилиты ROOT можно посмотреть результаты. Пример работы TestEm12 можно увидеть на Рис. 4 [2].

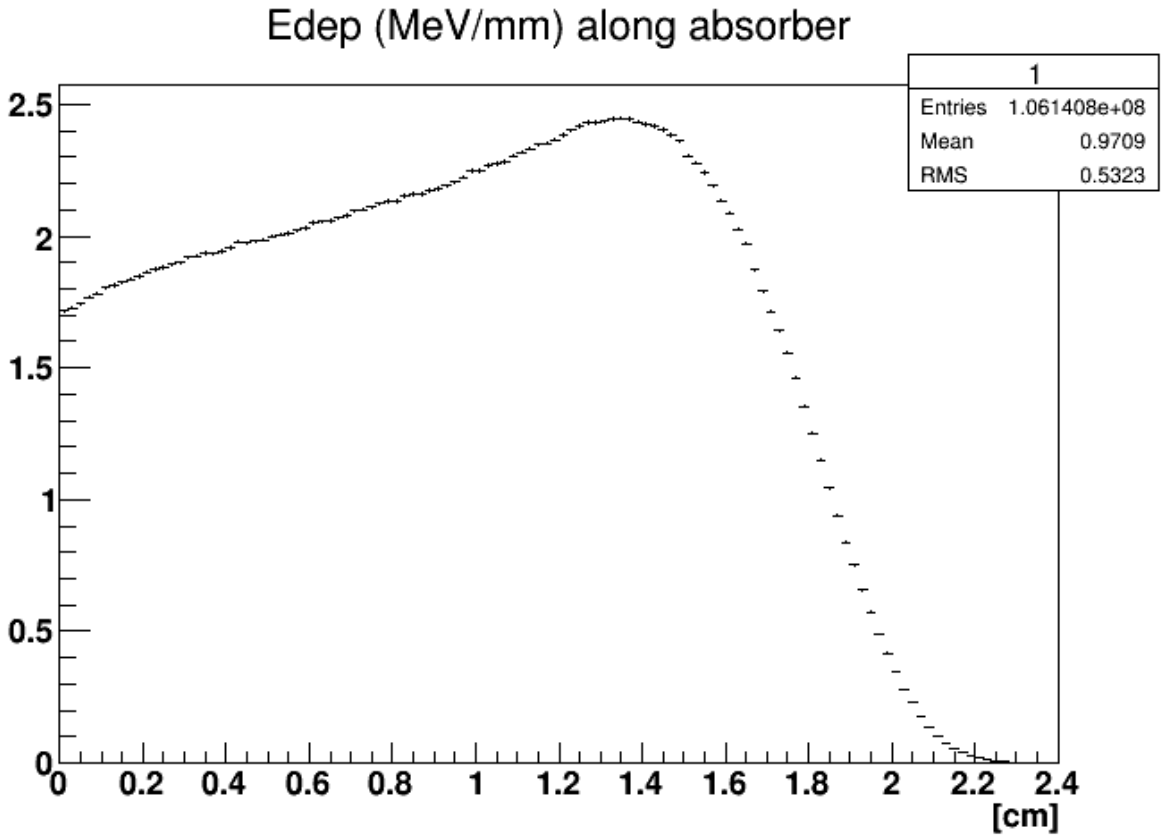

Рис. 4. Результат работы TestEm12 (сборка GCC без оптимизации) на 10^5 событий - потеря энергии (МэВ/мм) вдоль абсорбера. Гистограмма №1.

2.3.2. Параметры запуска

Код макроса запуска:

```
# $Id: run01.mac 78723 2014-01-20 10:32:17Z gcosmo $
#
# limit the step size from histos 1 and 8
#
/control/verbose 0
/run/verbose 0
/run/numberOfThreads 40
#
/testem/det/setMat G4_WATER
/testem/det/setRadius  3 cm
#
/testem/phys/addPhysics  local  # em physics
###/testem/phys/addPhysics  emlivermore # em physics
###/testem/phys/addPhysics  empenelope  # em physics
#
/run/initialize
#
/gun/particle e-
/gun/energy 4 MeV
#
/analysis/setFileName run01
/analysis/h1/set 1 120  0.   2.4 cm     #edep profile
/analysis/h1/set 3 100  0.    3. cm     #true track length
/analysis/h1/set 8 120  0.    1.2 none  #normalized edep profile
#
/testem/applyAutomaticStepMax true
#
/run/beamOn 100000
```

Значения в командах */run/numberOfThreads* и */rum/beamOn* меняются в зависимости от экспериментов.

Код макроса для отправки в SLURM:

```
#!/bin/sh
#SBATCH -p gpu
#SBATCH -w blade06
#SBATCH -N 1
#SBATCH -c 40
time ./TestEm12 run01.mac
```

Здесь значение параметра -c сопряжено со значением параметра numberOfThreads в макросе запуска приложения.

Исходный код примера в приложении 2.

2.3.3. Запуск приложения на CPU

Для каждого фиксированного количества событий на каждой сборке было произведено от 10 до 20 запусков. Полученные значения времени являются средним арифметическим от всех полученных результатов.

Тесты проводились на блэйдах группы gpu – blade04-blade07. На данных вычислительных узлах установлено по два процессора Intel Xeon Processor E5-2695 v2. На каждом процессоре 12 физических ядер, каждое физическое ядро имеет два логических. Итого на одном вычислительном узле 48 потоков.

10^5 событий

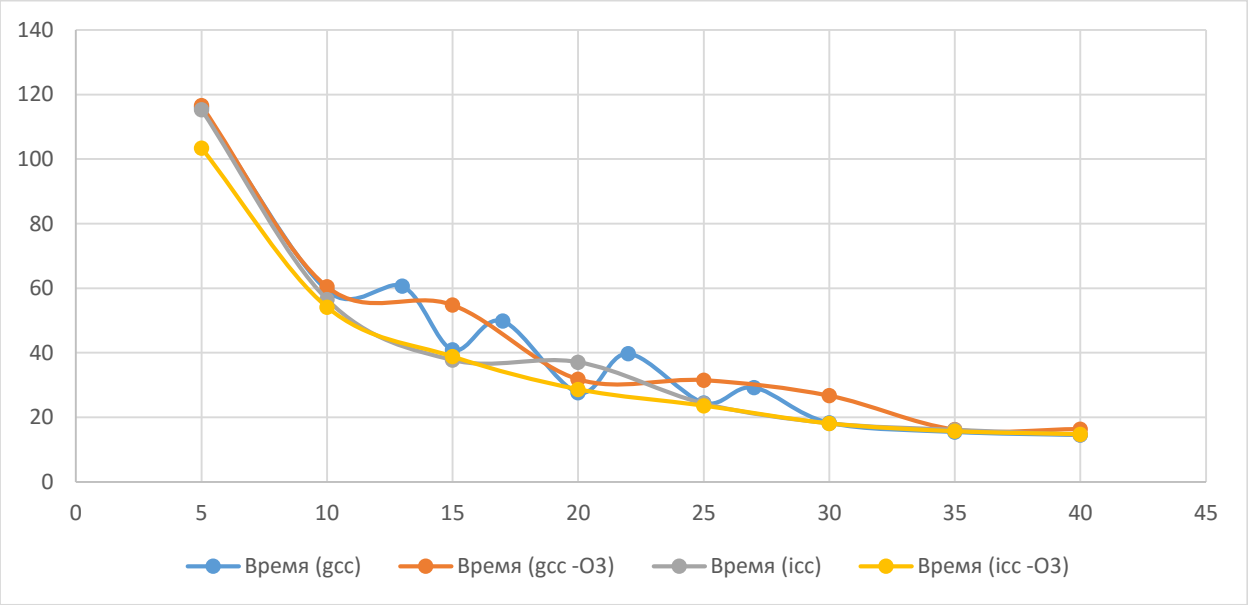

Рис. 5. Сравнение времени работы TestEm12 на 10^5 событий на GCC и ICC.

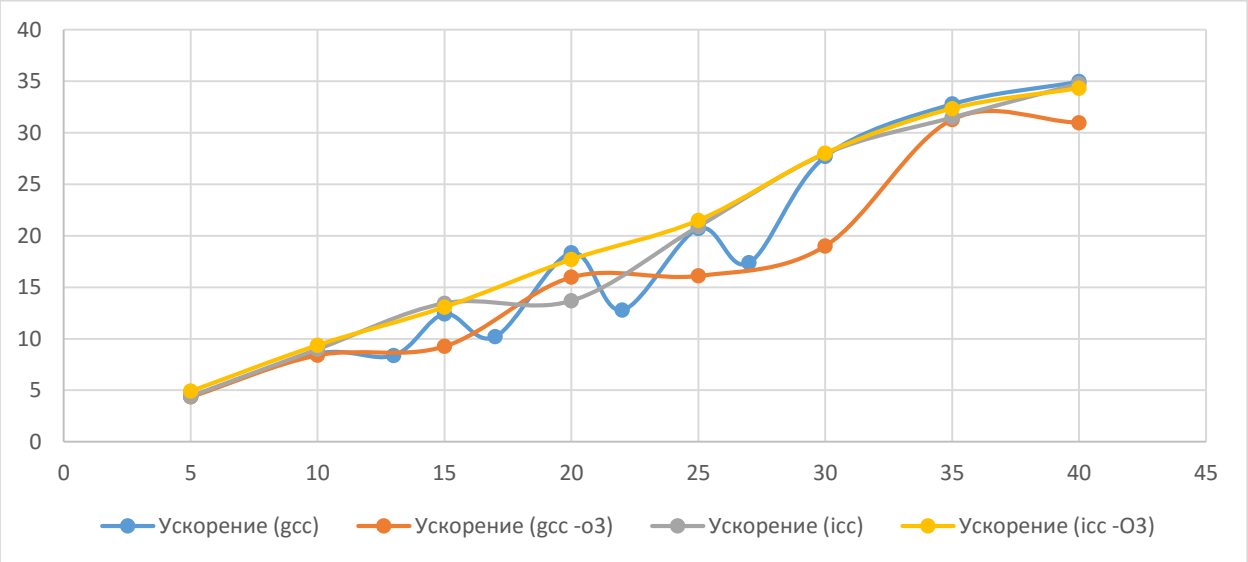

Рис. 6. Сравнение ускорения (speed-up) TestEm12 на 10^5 событий на GCC и ICC.

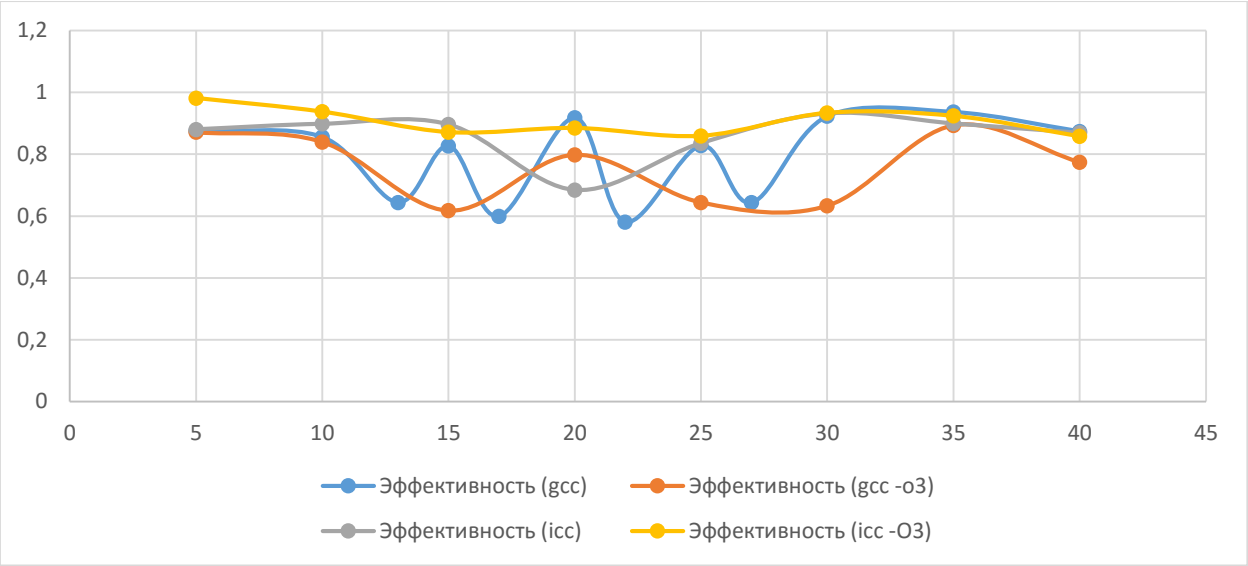

Рис. 7. Сравнение эффективности сборок TestEm12 на GCC и ICC на 10^5 событий.

На рисунках Рис. 5, Рис. 6 и Рис. 7 видно, что сборка на компиляторе GCC ведёт себя очень нестабильно на разном количестве потоков, в отличие от сборки на ICC. Проанализировав аппаратное обеспечение, было выдвинуто предположение, что данные скачки обусловлены использованием на процессорах Intel Xeon технологии Turbo Boost. Turbo Boost – технология Intel, позволяющая в автоматическом режиме увеличивать тактовую частоту ядер процессора в зависимости от нагрузки.

Для проверки этой гипотезы были проведены дополнительные тесты – с заниженной частотой и выключенным Turbo Boost. Результаты можно увидеть на Рис. 8.

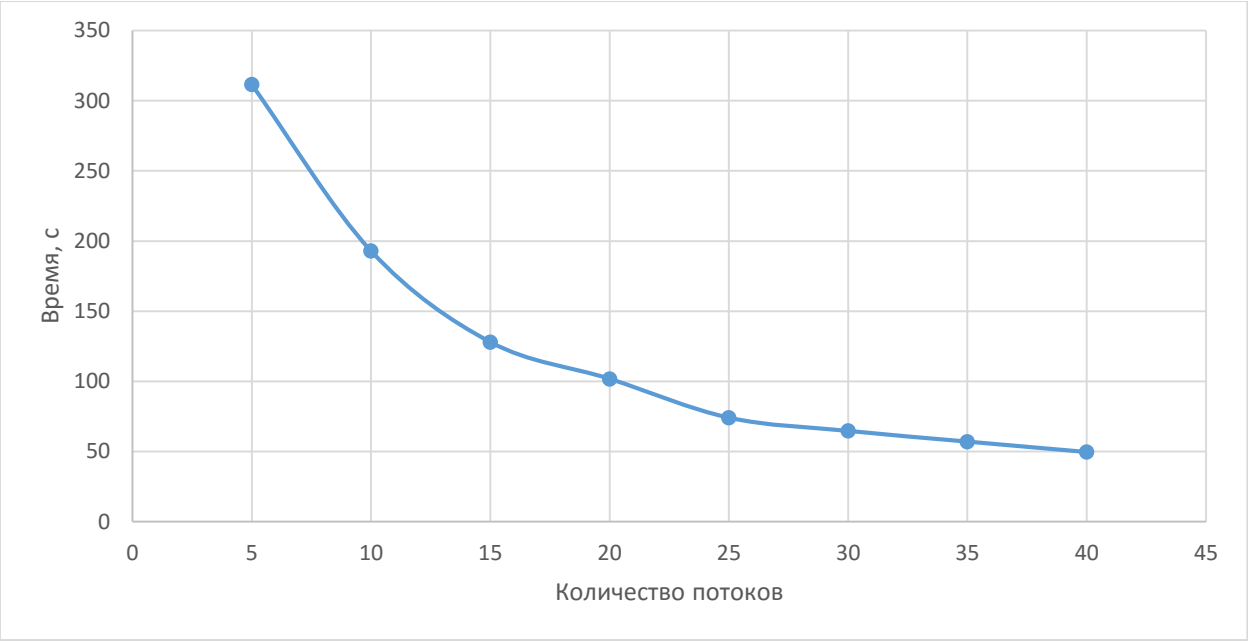

Рис. 8. Время работы сборки TestEm12 на GCC при выключенном TurboBoost и тактовой частоте ядер 1,2 ГГц.

На рисунке видно, что при выключенной технологии Turbo Boost график времени работы более плавный. Из выше описанного можно сделать вывод, что на таком количестве событий и при половинной загрузке процессора гораздо эффективней использовать компиляцию на ICC с флагом оптимизации -O3, чем другие. На большом количестве потоков разница сглаживается.

При этом можно выделить самую эффективную конфигурацию: сборка на ICC с флагом оптимизации -O3 с задействованием 30 потоков, однако выигрыш в скорости в сравнении с работой GCC совершенно незначительный: 18.1258 с у ICC -O3 против 18.3373 с у GCC. Преимущество чуть более одного процента.

Также были проведены тесты со сборкой GCC -O2 и ICC -O2, и обе показали результаты, идентичные сборкам с флагом -O3. Ввиду этого на графиках они отсутствуют.

10^6 событий

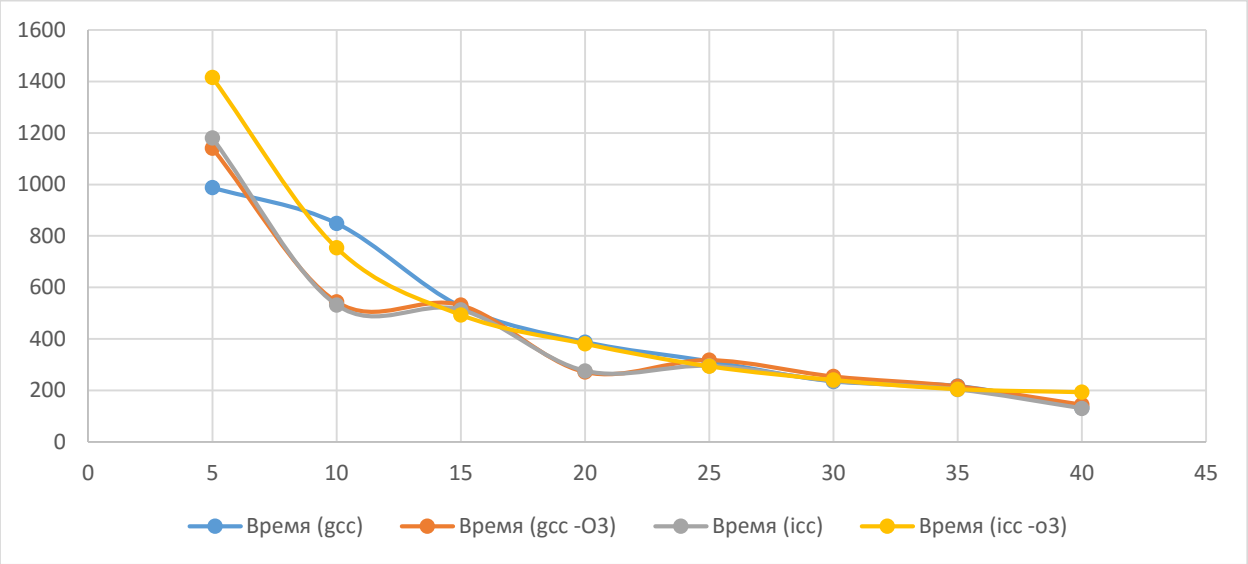

Рис. 9. Сравнение времени работы сборок TestEm12 на GCC и ICC на 10^6 событий.

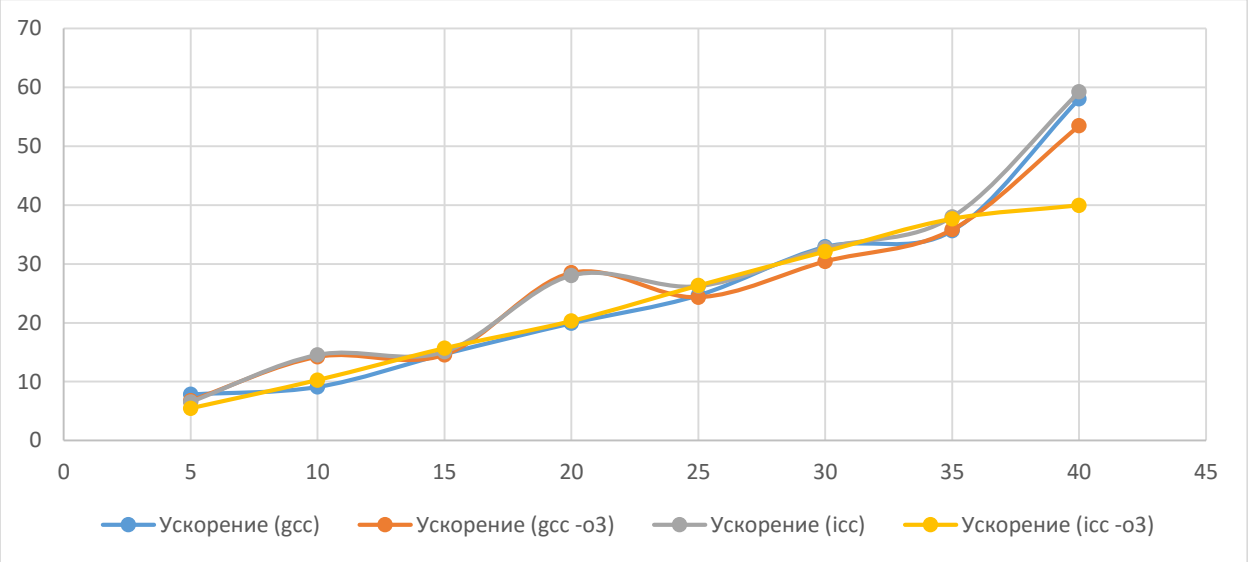

Рис. 10. Сравнение ускорения (speed-up) TestEm12 на 10^6 событий на GCC и ICC.

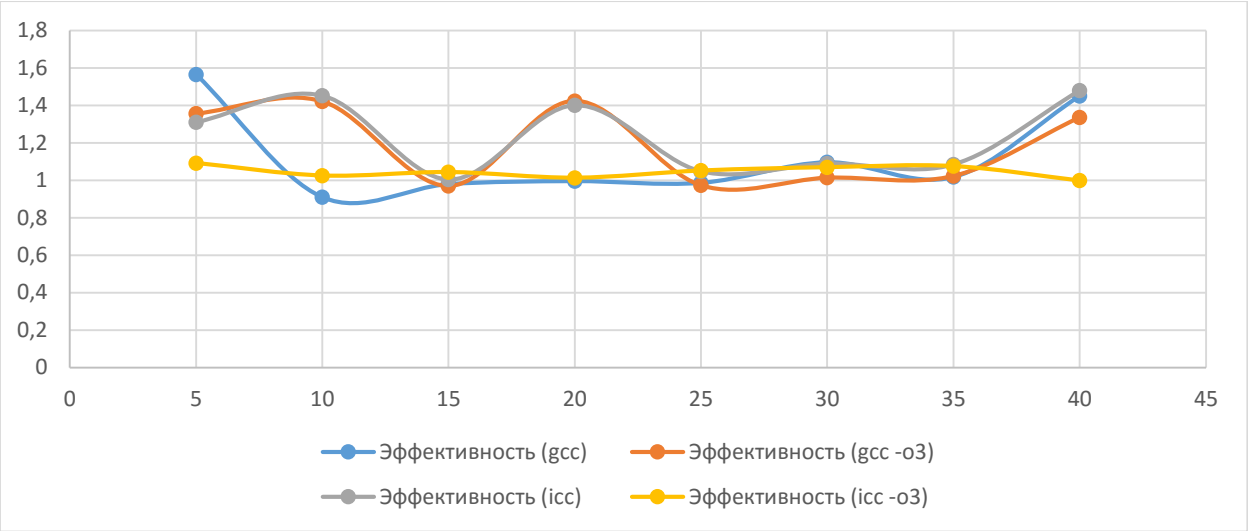

Рис. 11. Сравнение эффективности сборок TestEm12 на GCC и ICC на 10^6 событий.

На рисунках Рис. 9, Рис. 10 и Рис. 11 видно, что с увеличением количества событий разница между сборками на разных компиляторах почти сгладилась, однако слабые скачки на небольшом количестве потоков всё ещё остались. При этом, в отличие от прошлого эксперимента, максимальную эффективность показала сборка на ICC без оптимизации на 40 потоках. Выигрыш во времени получился снова незначительным. Сборка на ICC показала результат 130.6269 секунд на 40 потоках, а GCC – 133.2989 секунд. Выигрыш чуть больше 2%.

10^7 событий

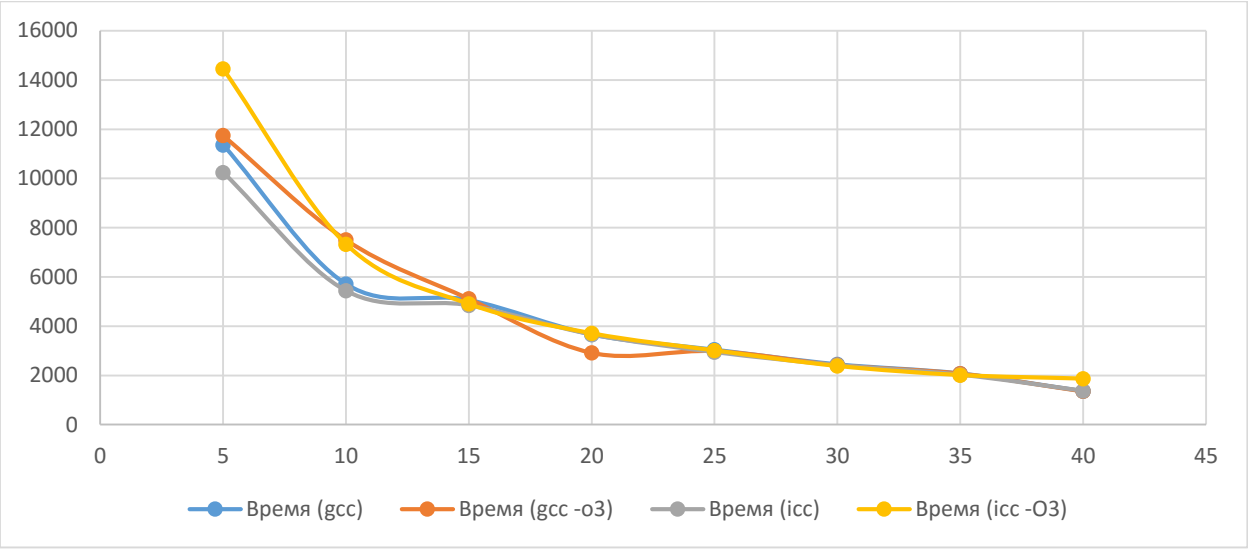

Рис. 12. Сравнение времени работы сборок TestEm12 на GCC и ICC на 10^7 событий.

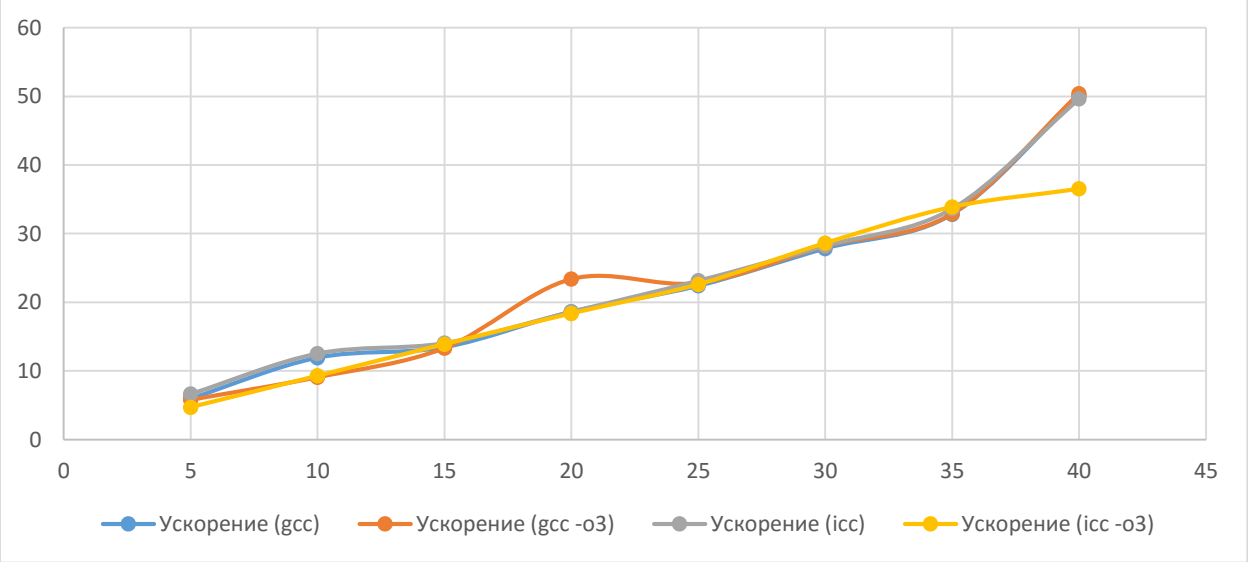

Рис. 13. Сравнение ускорения (speed-up) сборок TestEm12 на GCC и ICC на 10^7 событий.

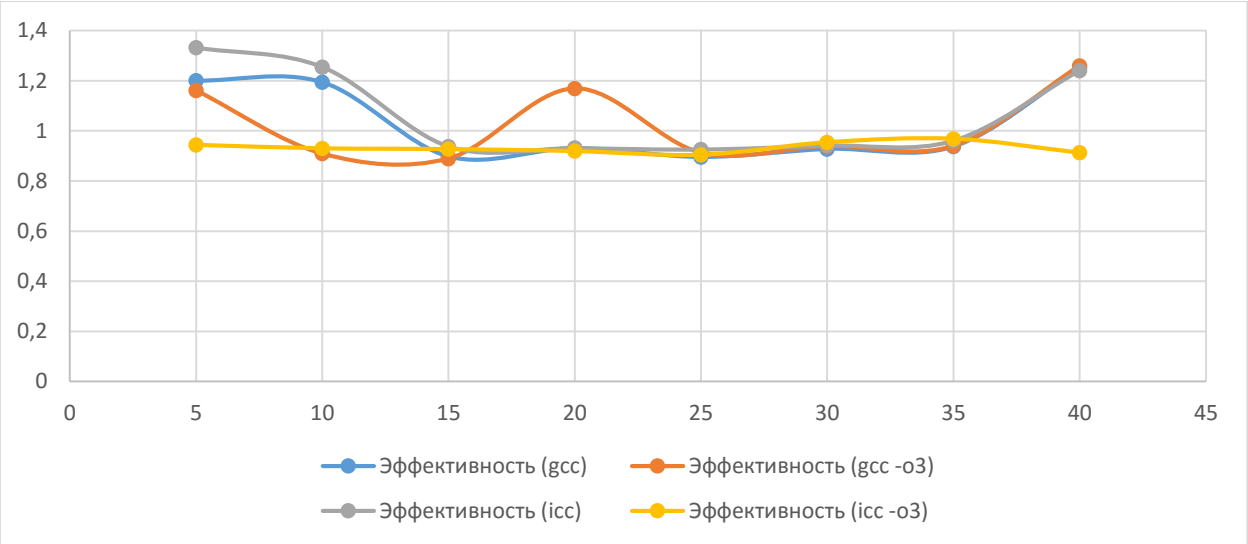

Рис. 14. Сравнение эффективности сборок TestEm12 на GCC и ICC на 10^7 событий.

В этом эксперименте, как видно по Рис. 12, Рис. 13 и Рис. 14, получились результаты, похожие на те, что были получены на предыдущих тестах: разница в работе сборок почти полностью сглаживается, начиная с 25 потоков. И снова сборка ICC -O3 показала худшие результаты в сравнении с другими сборками на 40 потоках. При этом время работы на ICC, GCC и GCC -O3 примерно одинаковое, но всё же последняя показала лучший результат – 1353.057 секунд против 1373.989 на ICC. Выигрыш во времени у GCC перед ICC - ~1.5%.

Были произведены замеры времени инициализации, которые показали незначительное время для любого количества потоков (~1 с).

2.3.4. Сборка и запуск приложения на сопроцессоре

Сопроцессор Intel Xeon Phi – ускоритель, построенные на архитектуре Intel MIC (с англ. Many Integrated Core Architecture – многоядерная интегрированная архитектура). Для тестов на кластере HybriLIT был выбран блэйд с установленным на нём Intel Xeon Phi 7120P. Данная модель имеет 61 физическое ядро, каждое из которых обладает четырьмя потоками благодаря технологии hyper-threading (с англ. гиперпоточность).

Особенностью работы сопроцессора Xeon Phi является то, что он не требует дополнительных манипуляций с кодом, написанным для CPU. Так, можно произвести кросс-компиляцию написанного для центрального процессора кода, затем исполняемый файл можно запустить в native-режиме на сопроцессоре. Native-режим – запуск приложения непосредственно на сопроцессоре, в отличие от Offload, который предполагает запуск на CPU и взаимодействие с сопроцессором с помощью соответствующих команд в коде.

Кросс-компиляция происходила следующим образом:

1. Настройка переменных окружения:

```
$ export CC=icc
$ export CXX=icpc
$ export LD=/usr/linux-k1om-4.7/bin/x86_64-k1om-linux-ld
$ export AR=/usr/linux-k1om-4.7/bin/x86_64-k1om-linux-ar
$ export LDFLAGS=-mmic
$ export CXXFLAGS=-mmic
$ export CFLAGS=-mmic
```

2. Подготовка файла mic-toolchain.cmake:

```
SET(CMAKE_SYSTEM_NAME Linux)
SET(CMAKE_SYSTEM_VERSION 1)

# описание кросс-компиляции
SET(CMAKE_C_COMPILER icc)
SET(CMAKE_CXX_COMPILER icpc)
SET(CMAKE_FIND_ROOT_PATH  /opt/intel/composer_xe_2013_sp1.2.144)

# search for programs in the build host directories
```

```
SET(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
# for libraries and headers in the target directories
SET(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
SET(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

3. Кросс-компиляция пакета:

```
$ cmake -DGEANT4_BUILD_MULTITHREADED=ON \
        -DGEANT4_USE_SYSTEM_EXPAT=OFF \
        -DGEANT4_INSTALL_DATA=OFF -DGEANT4_INSTALL_DATADIR=<путь к data-файлам Geant4>/data \
        -DCMAKE_C_COMPILER=${CC} -DCMAKE_CXX_COMPILER=${CXX} -DCMAKE_LINKER=${LD} -DCMAKE_AR=${AR} \
        -DCMAKE_TOOLCHAIN_FILE=<путь файлу toolcahin>/mic-toolchain-file.cmake \
        -DBUILD_SHARED_LIBS=OFF -DBUILD_STATIC_LIBS=ON \
       <путь к исходным файлам Geant4>
```

4. Кросс-компиляция приложения:

```
$ cmake -DCMAKE_LINKER=${LD} -DCMAKE_AR=${AR} \
        -DCMAKE_TOOLCHAIN_FILE=<путь к файлу toolchain>/mic-toolchain-file.cmake \
        -DGeant4_DIR=<путь к билду Geant4 из первого шага> \
        <путь к исходникам примера>
```

После компиляции папку с приложением необходимо скопировать на сопроцессор командой scp.

Тесты на сопроцессоре были единичные, но с относительно мелким шагом (по 10 потоков на максимальные 240). Приложение не было тестировано на максимальной загрузке (244 потока), чтобы не получать погрешность из-за работы операционной системы.

Ввиду отсутствия на сопроцессоре менеджера очередей SLURM все запуски происходили в интерактивном режиме.

10^5 событий

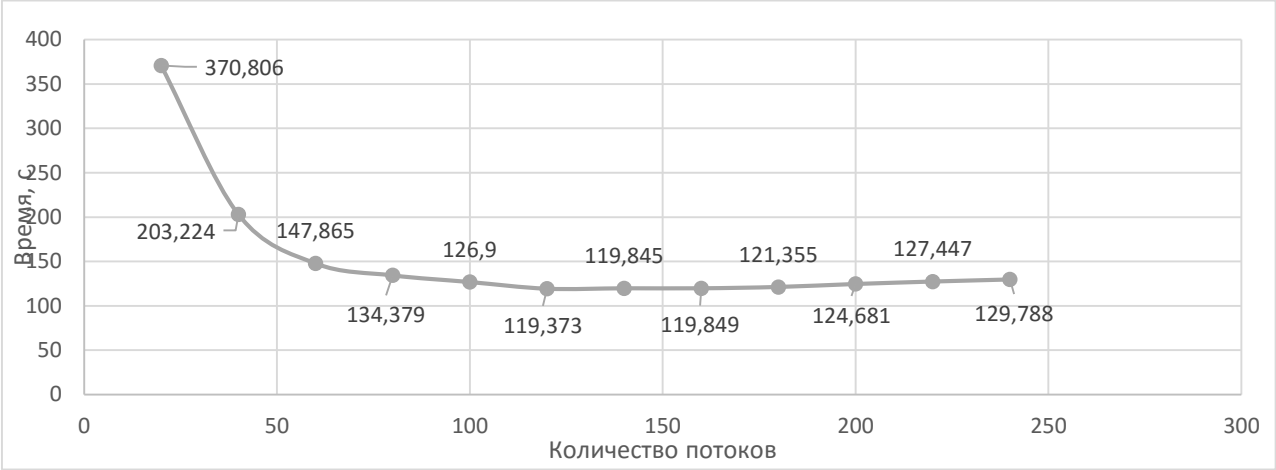

Рис. 15. Время работы TestEm12 на сопроцессоре Intel Xeon Phi на 10^5 событий.

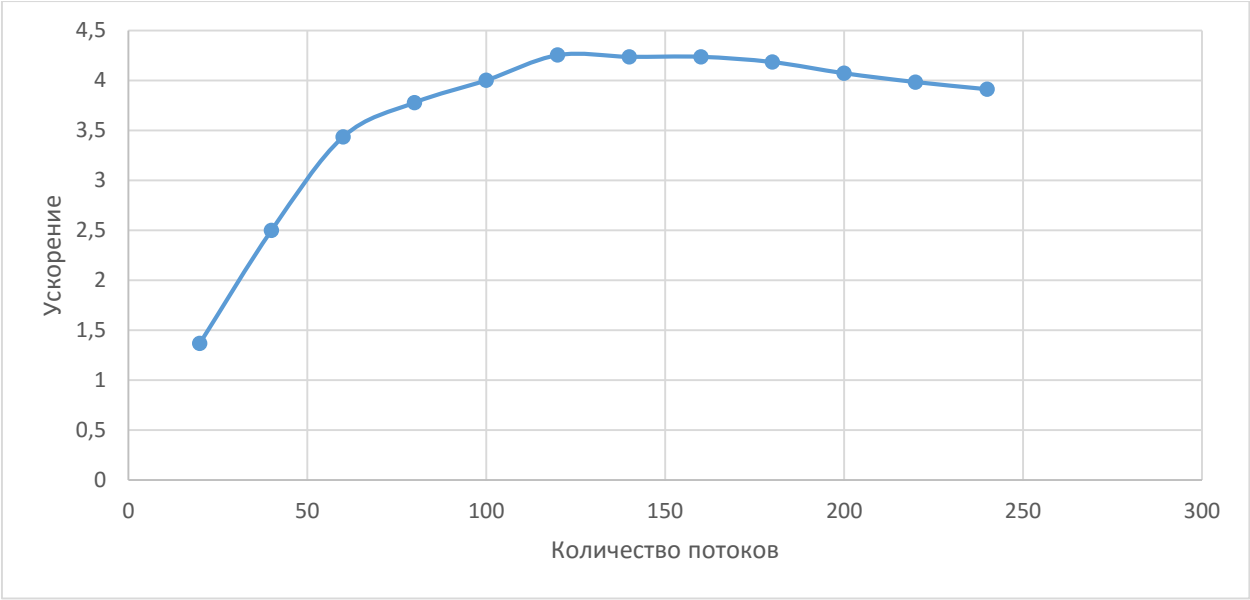

Рис. 16. Ускорение TestEm12 на Intel Xeon Phi на 10^5 событий.

Тесты на сопроцессоре показали очень слабые результаты. Так, на Рис. 15 видно, как уменьшение времени работы с ростом количества потоком замедляется почти сразу и останавливается примерно на 120 потоках, что более отчётливо видно на Рис. 16 с графиком ускорения.

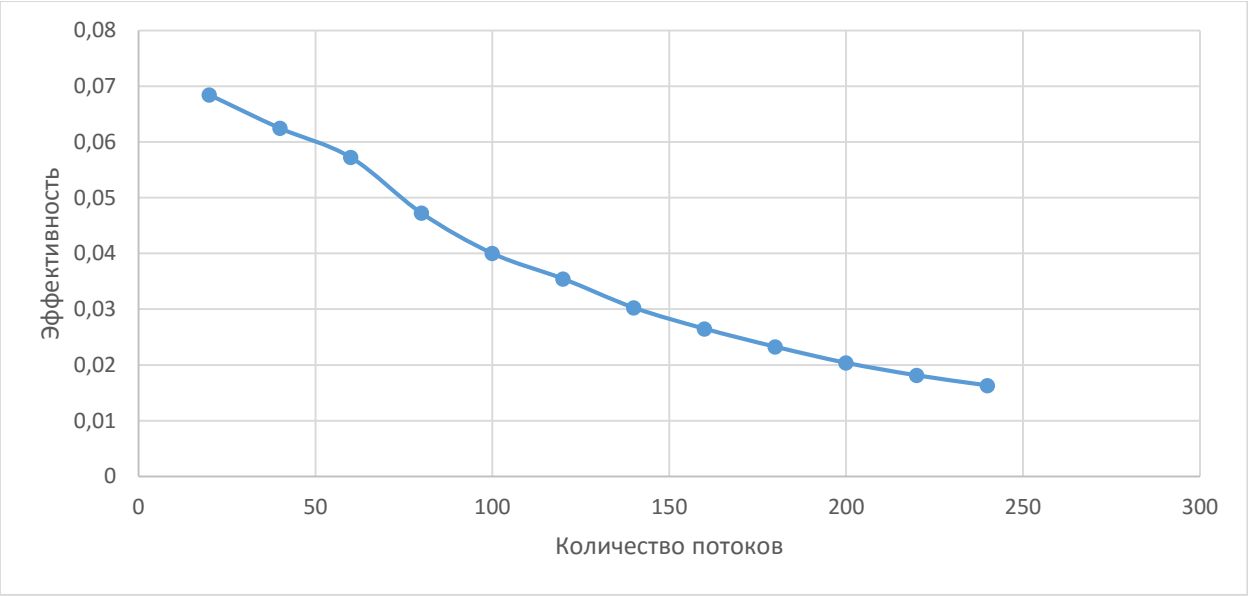

Рис. 17. Эффективность работы TestEm12 на Intel Xeon Phi на 10^5 событий.

Эффективность при этом, как показано на Рис. 17, всегда падает. Лучший результат достигнут при работе на 120 потоках – 119.373 с, что хуже, чем результат работы на CPU с использованием пяти потоков.

10^6 событий

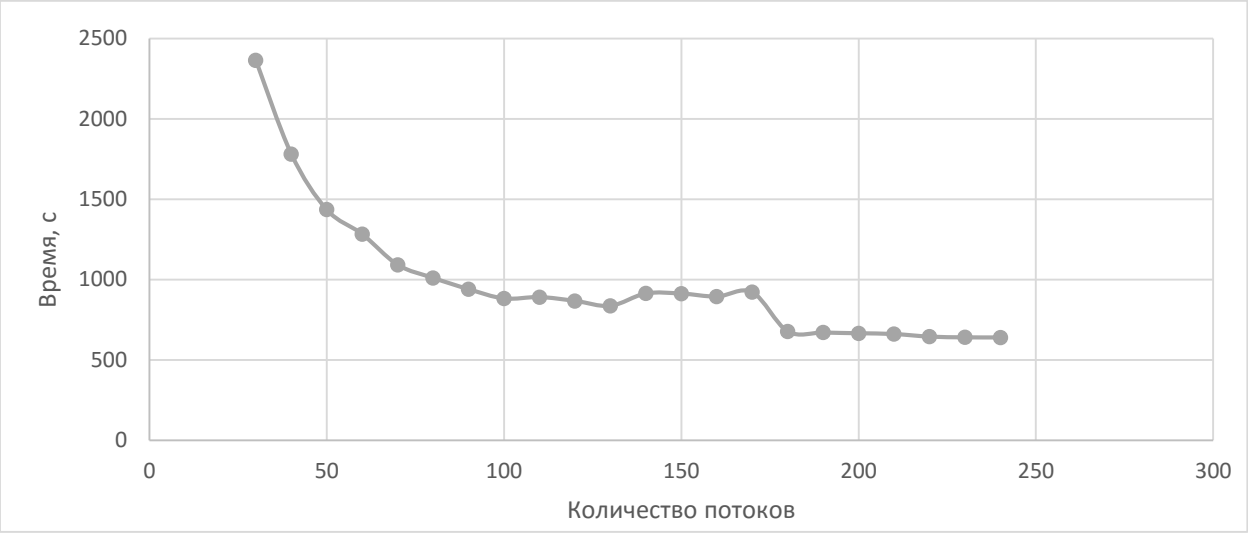

Рис. 18. Время работы TestEm12 на Intel Xeon Phi на 10^6 событий.

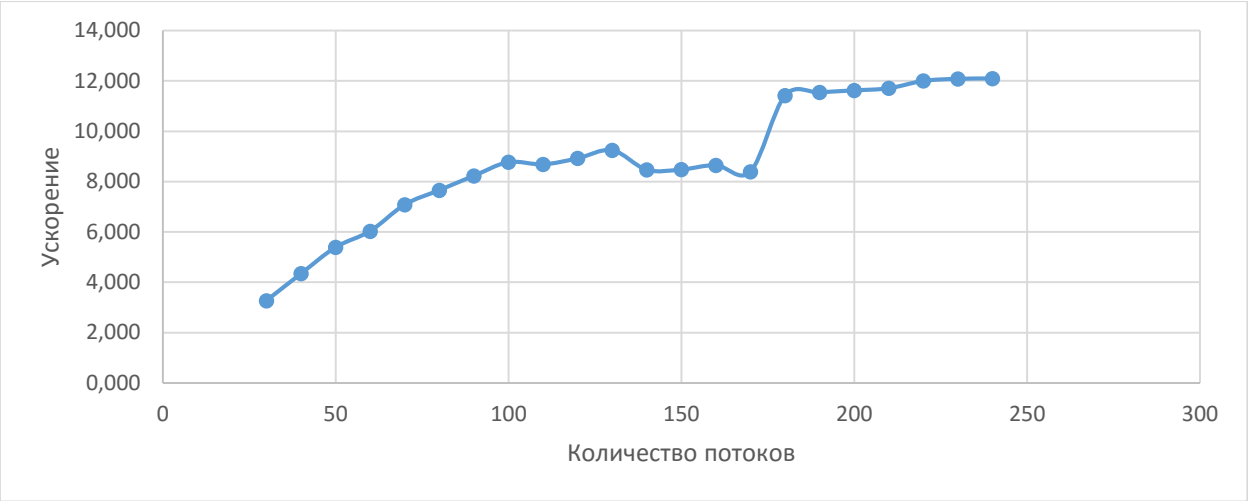

Рис. 19. Ускорение TestEm12 на Intel Xeon Phi на 10^6 событий.

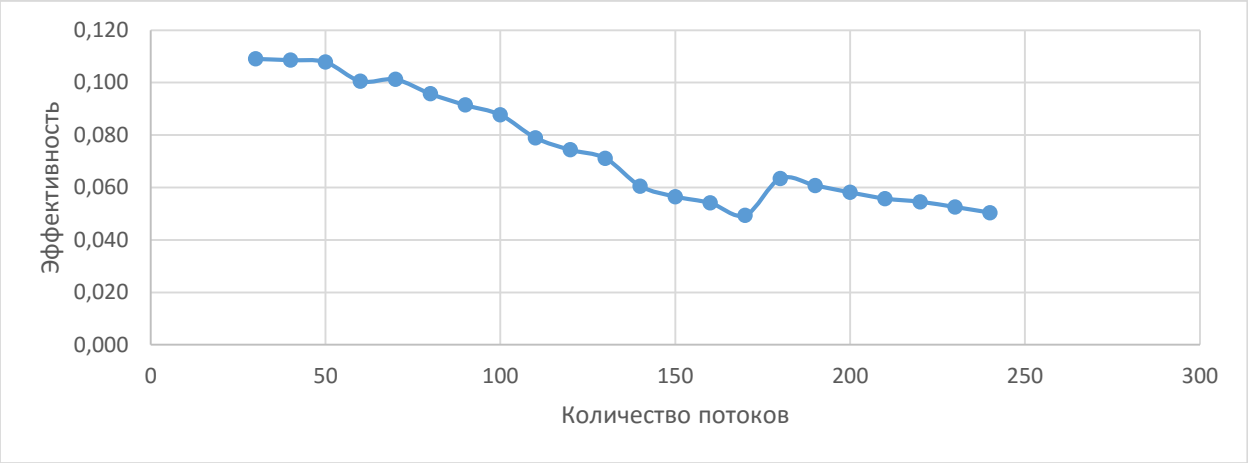

Рис. 20. Эффективность работы TestEm12 на Intel Xeon Phi на 10^6 событий.

На большем количестве событий сопроцессор показал лучшие результаты, но динамика осталась та же – постоянное падение эффективности (Рис. 20), лучшее время достигнуто при задействовании 240 потоков – 639.46 с, однако близкий результат получен и на 180 потоках – 677.549 с (Рис. 18, Рис. 19), хуже всего на менее чем 6%. И тем не менее, это время гораздо хуже, чем на CPU – быстрее, чем 5 потоков (1180.367 с на ICC), но медленнее, чем 10 (532.3972 с на ICC).

10^3 событий

Были проведены тесты для оценки времени загрузки потоков. Для этого было выбрано маленькое количество событий (всего 1000). Результаты тестов

представлены на Рис. 21.

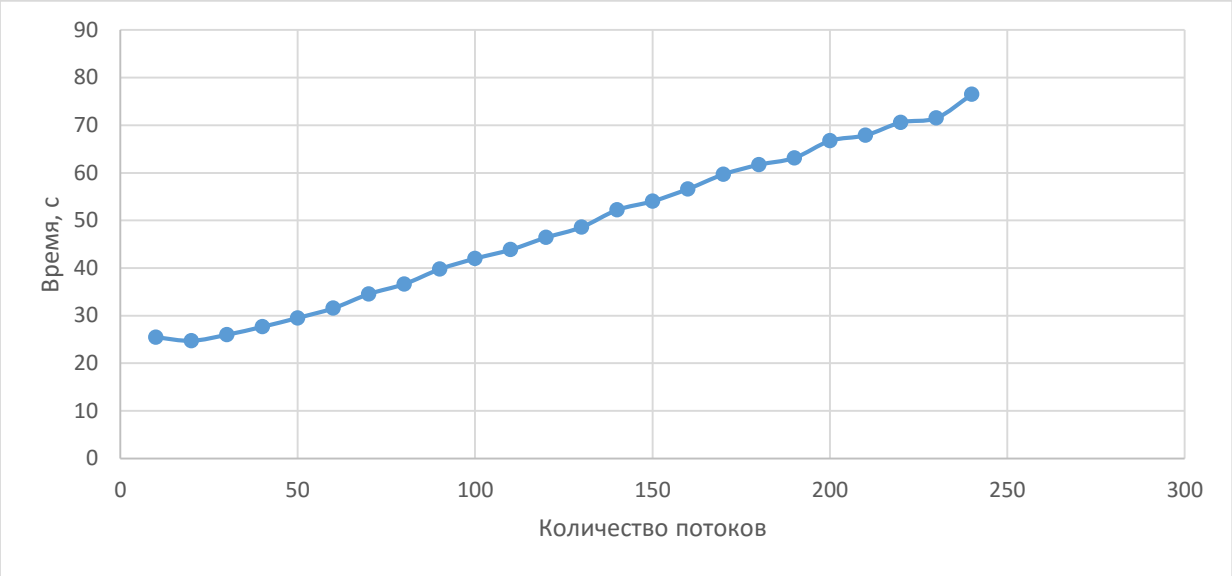

Рис. 21. Время работы TestEm12 на Intel Xeon Phi на 1000 событий.

На графике времени видно, что загрузка потоков занимает огромное время на сопроцессоре: 25 секунд на 10 потоках и более 70 секунд на почти максимальной нагрузке.

2.4. Тестирование сложного примера

2.4.1. Описание примера

Выбор сложного примера для тестирования пал на Underground Physics (UP) в папке *examples/advanced*. Данный пример представляет собой подземный эксперимент тёмной материи.

Задача: симуляция жидкой ксеноновой клетки и запись мерцания света от взаимодействий как срабатываний фотоэлектронного умножителя (ФЭУ). Выходные данные записываются в ASCII файл для дальнейшего анализа.

Описание геометрии

«Пещера» размером 5.18м*7.38м*3.28м с бетонными стенами определена как World Volume (объём мира). Геометрия лаборатории включает

в себя столы, шкафы, двери и окна. В центре пещеры стальной вакуумный сосуд, содержащий жидкий и газообразный ксенон. Внуренняя конструкция сосуда в точности повторяет реальную конструкцию детектора Тёмной Материи, что обеспечивает сравнимость экспериментов. Активный объём детектора определён несколькими металлическими кольцами, дополненными покрывающим зеркалом и трубкой ФЭУ, погружённой в жидкость. Присутствуют также две сетки и термализующий медный экран. Интерфейс газа/жидкости расположен в 6 мм от поверхности зеркала. Источник калибровки Am^{241} (америций) подвешен в одной из сеток в жидком состоянии, над трубкой ФЭУ. Схематичное представление детектора изображено на Рис. 22.

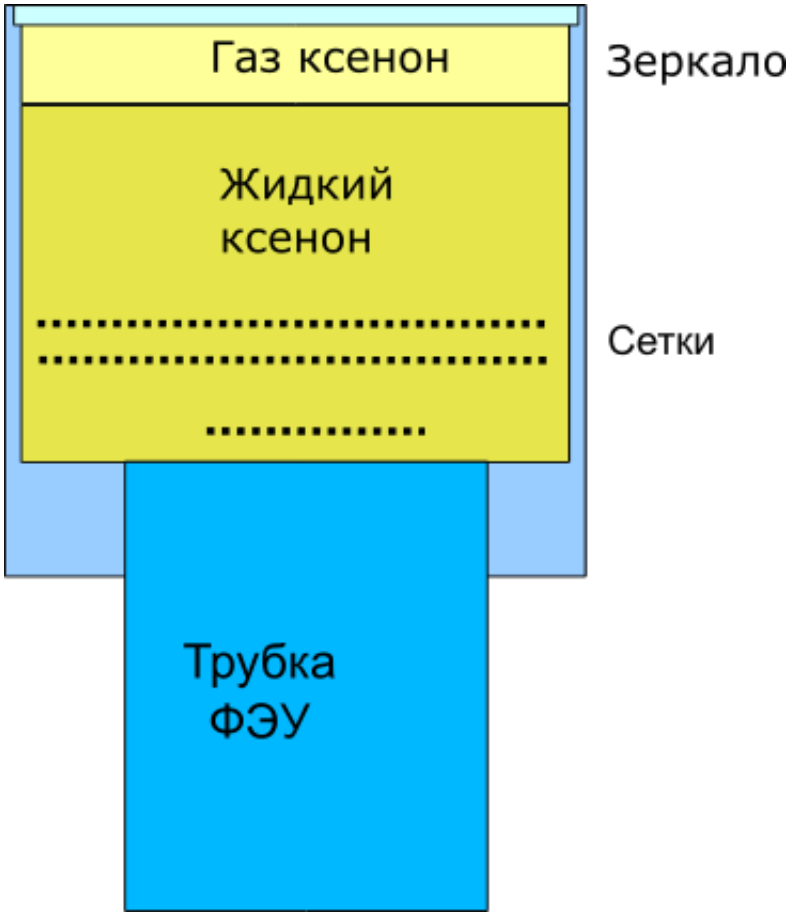

Рис. 22. Схематичное изображение детектора в подземном эксперименте Тёмной материи.

Выходные файлы

Файл с зарегистрированными срабатываниями (*hits.out*) содержит информацию в формате ASCII:

1. Evt # - номер события;
2. Etot, MeV – энергия, оставленная в жидком ксеноне;
3. LXe hits – количество срабатываний в жидком ксеноне;
4. LXeTime, ns – время первого срабатывания в жидком ксеноне (в наносекундах);
5. PMT hits – количество срабатываний в трубке ФЭУ (фотокатод);
6. PmtTime, ns – среднее время срабатывания в трубке ФЭУ относительно LXeTime;
7. First hit – первая частица, сработавшая в жидком ксеноне;
8. Flags – частицы, способствующие потере энергии;
9. Seeds – начальные значения событий, в которых произошло срабатывание.

Файл с данными от ФЭУ (файл *pmt.out*) содержит позиции срабатываний фотонов на трубке ФЭУ (переписывается на каждом событии):

Hit# X, mm Y, mm Z, mm

2.4.2. Запуск приложения

Компиляция и запуск приложения осуществляется аналогично предыдущему примеру. Макрос запуска и SLURM-скрипт выглядят похожим образом. Исходный код примера в приложении 3.

Тесты были проведены на 8 и 9 блэйдах с установленными на них Intel Xeon Processor E5-2695 v3 (в отличие от процессоров второй версии, здесь на 5 Мб больше кэш, на 2 физических ядра больше, на 0,1 ГГц выше максимальная тактовая частота и т.д., более подробно на сайте Intel).

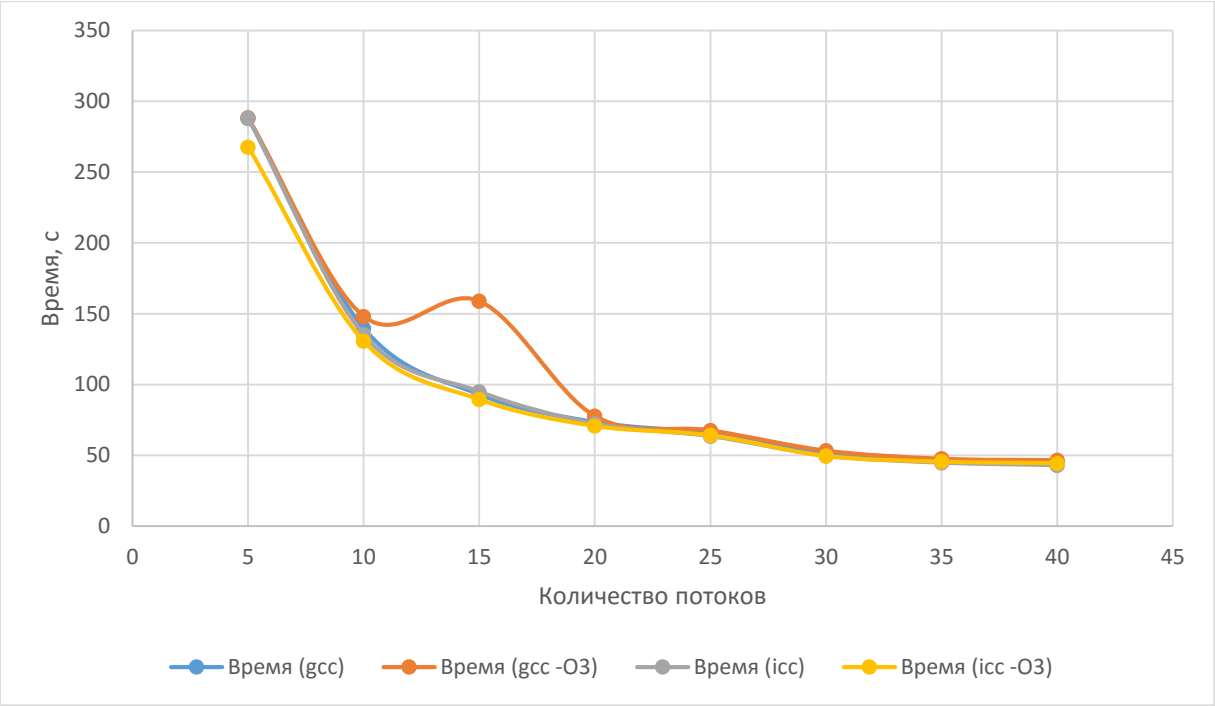

Рис. 23. Сравнение времени работы UP на 10^5 событий на GCC и ICC.

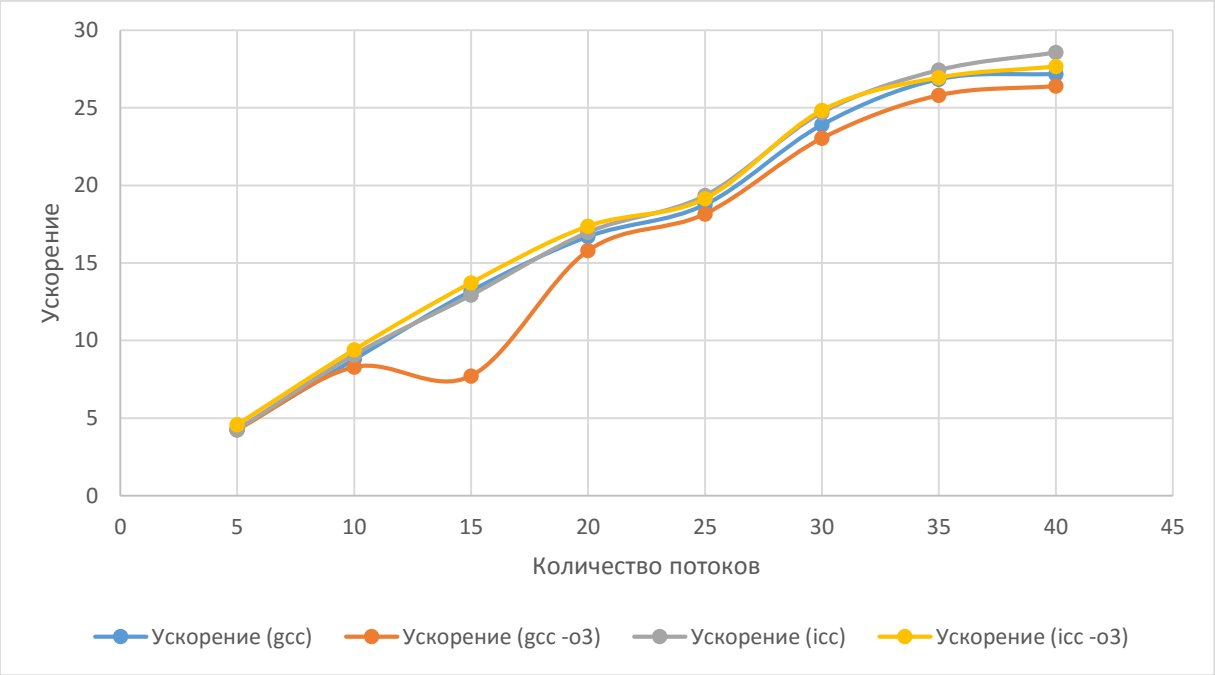

Рис. 24. Ускорение (speed-up) UP на GCC и ICC на 10^5 событий.

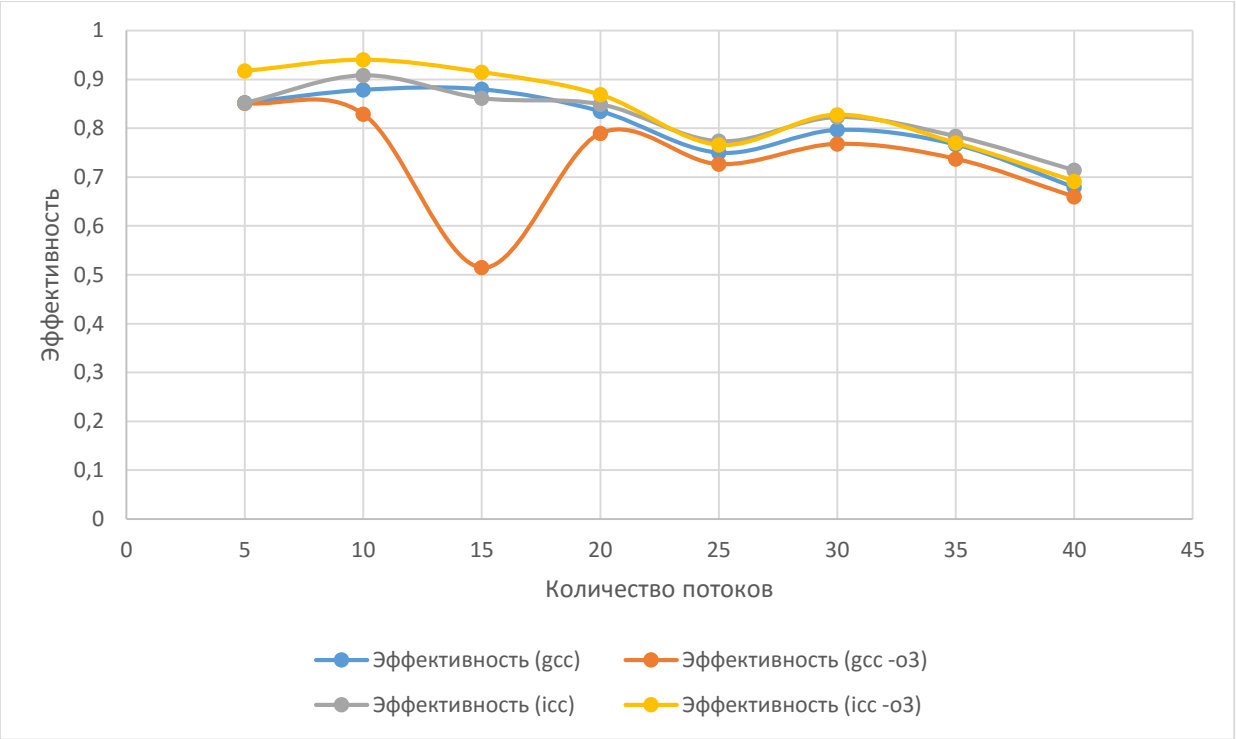

Рис. 25. Эффективность работы UP на 10^5 событий на GCC и ICC.

В отличие от TestEm12, пример Underground Physics показал более схожие результаты работы сборок на GCC и ICC, как видно на Рис. 23, Рис. 24 и Рис. 25. Три из четырёх сборки показали схожие результаты с незначительным отклонением. Исключение – сборка на GCC с флагом оптимизации -O3, которая при задействовании 15 потоков показала почти на 78% худший результат в сравнении с другими сборками. Максимальную эффективность показали сборки ICC (49.732 с) и ICC -O3 (49.4424 с) на 30 потоках. Выигрыш в сравнении с работой сборки на GCC (51.3761 с) составил всего около 4%.

Были проведены замеры времени инициализации. Для любого количества потоков время варьировалось от 10 до 13 секунд.

Выводы

Исследование показало, что на простом примере в среднем компилятор Intel показывает лучшие результаты в сравнении с GNU, однако разница эта незначительна – порядка нескольких процентов, и с ростом количества событий сглаживается. При этом максимальная эффективность замечена на 30-35 задействованных потоках. Данные результаты говорят об отсутствии необходимости использовать коммерческий компилятор Intel ввиду неощутимой разницы в работе в сравнении со свободным GNU. Некоторый видимый выигрыш во времени есть лишь на задействовании половины мощности центрального процессора.

При работе с более сложным приложением Geant4 разница между сборками на компиляторах Intel и GNU практически стёрлась, что является дополнительной причиной не использовать лишний раз коммерческий компилятор.

Также проведены эксперименты с использованием флага оптимизации -ON со значениями N, равные 2 и 3. Сборка на ICC в обоих случаях почти не показала изменений либо получила небольшой прирост к скорости. В случае с GCC же на примере TestEm12 прирост имел место, но на Underground Physics на 15 потоках был явный провал во времени, затем же, с увеличением количества потоков, сборка показывала несколько лучшие результаты в сравнении с другими.

Помимо этого, пример TestEm12 был запущен на сопроцессоре Intel Xeon Phi на 10^5 и 10^6 событий. К сожалению, сопроцессор показал низкие результаты в сравнении с работой CPU Intel Xeon – лучшие результаты получились ниже результатов CPU даже при работе на половинной мощности. В связи с этим можно сделать вывод, что для работы пакета Geant4 сопроцессор Intel Xeon Phi неэффективен. Возможно, лучшие результаты во времени работы могут быть достигнуты в режиме Offload с помощью

экспериментов в изменении кода и распределении вычислений между ЦП и ускорителем.

Заключение

В рамках сотрудничества между СПбГУ и ЛИТ ОИЯИ было проведено исследование эффективности работы гетерогенного кластера HybriLIT с пакетом Geant4, в результате чего найдены решения, дающие как прирост к скорости вычислений, так и потенциальную экономическую выгоду. Ввиду постройки проекта NICA данное исследование особенно актуально, однако для получения более значительных результатов необходима оптимизация инструментария на уровне исходных кодов.

В ходе работы были проведены тесты простого и сложного приложений пакета Geant4 и сделаны замеры времени выполнения при использовании разных компиляторах, а также на разном оборудовании (центральный процессор и сопроцессор). Полученные результаты переданы команде HybriLIT в ЛИТ ОИЯИ для проведения дальнейшей экспертизы.

Список литературы

1. А.Н. Сисакян «Избранные лекции по физике частиц», «Развитие физики высоких энергий и ускорителей». http://wwwinfo.jinr.ru/publish/Books/sisakian/content_sis_rus.html
2. Адам Г., Беляков Д.В., Валя М. и др. Особенности программно-аппаратной среды гетерогенного кластера вычислительного кластера HybriLIT // Информационно-телекомуникационные технологии и матмоделирование, М.: РУДН, 2016. С. 197-198.
3. Адам Г., Вальова Л., Валя М. и др. Информационная среда гетерогенного кластера HybriLIT // HybriLIT // Информационно-телекомуникационные технологии и матмоделирование, М.: РУДН, 2016. С. 199-200.
4. Бекман И.Н. Курс лекция «Ядерная физика». http://profbeckman.narod.ru/YadFiz.files/L1.pdf
5. Борн М. Атомная физика. М.: Мир, 1965. с. 78-82.
6. Гетерогенный кластер HybriLIT. http://hybrilit.jinr.ru/
7. Дрёмин И.М. Физика на Большом адронном коллайдере // Успехи физических наук, 2009. Т. 179, №6. С. 571-579.
8. Ишханов Б.С., Кэбин Э.И. Физика ядра и частиц, XX век. http://nuclphys.sinp.msu.ru/introduction/
9. Левин А. Частица-призрак: нейтрино // Популярная механика, 2010, №3.
10. Лекции МФТИ-2016, «Пакет Geant4. Основные понятия”. http://geant4.jinr.ru/
11. Леонтьев В.В., Орлов И.А. Задачи раздела “Информационные методы в физике высоких энергий», часть 2. 2013.
12. Перкинс Д. Введение в физику высоких энергий. М.: Энергоатомиздат, 1990. с. 5-7.
13. Скорохватов М.Д. Нейтринная геофизика - первые шаги // Природа, 2012, №3, с. 13-17.

14. Тарасов Л. Открытие нейтрона // Квант, 1979, №5, с. 8-14.
15. Телеснин Р.В. Молекулярная физика. Изд. 2-е, доп. М.: Высшая школа, 1973. с. 15-16.
16. Широков Е.В. «Физика нейтрино сверхвысоких энергий». http://nuclphys.sinp.msu.ru/neutrino/uen/index.html
17. Элементарный учебник физики / Под реда. академика Г.С. Ландсберга. Том 1. Изд. 1-е, М.: Наука, 1985. с. 452-453.
18. Agostinelli S., Allison J., Amako K. and others. Geant4 – simulation toolkit // Nucl. Instr. And Methods in Phys. Researc Section A: Acc., Spectr., Detectors and Ass. Equipment, 2003, vol. 506, Issue 3, p. 250-303.
19. Example TestEm12. http://geant4.web.cern.ch/geant4/UserDocumentation/Doxygen/examples_doc/html/ExampleTestEm12.html
20. G.N. Flerov. On the synthesis of element 105 // Nuclear Physics A, 1970, Vol. 160, Issue 1, p. 181-192.
21. Geant4 Software Installation. http://geant4.web.cern.ch/geant4/collaboration/working_groups/softman/training/installation.pdf
22. Installation guide for Geant4. http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/InstallationGuide/
23. Perricone M. GEANT4 the physics simulation toolkit // symmetry, 2005. Vol. 2, Issue 9, p. 20-23.
24. Physics Reference Manual. http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/PhysicsReferenceManual/fo/PhysicsReferenceManual.pdf
25. Schildknecht D. Problems with Ultrahigh-energy Neutrino Interactions. http://arxiv.org/pdf/1411.0498v1.pdf

26. Schweitzer P., Cipière S., Dufaure A. Performance Evaluation of Multithreaded Geant4 Simulations Using an Intel Xeon Phi Cluster // Scientific Programming, 2015, vol. 2015, Article ID 980752.
27. Shun Zhou «Theoretical Results on Neutrinos». http://arxiv.org/pdf/1511.07255v1.pdf
28. User Requirements Document. http://geant4.web.cern.ch/geant4/OOAandD/URD.pdf
29. User's Guide for Toolkit Developers. http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForToolkitDeveloper/html/index.html - Geant4

Приложение 1. Аппаратное обеспечение кластера HybriLIT

Storage Blade (6x 2,5" SAS2/SATA), blade01

2xCPU IvyBridge-EP E5-2695v2 12C 115W 2.40G 30M 8.00GT/sec LGA2011;

8x16GB DDR3-1600 ECC REG DIMM 1.5v;

6xHDD 2,5" 1,2TB SAS2 64MB 10000RPM 0B25168 HGST

Phi 7120P 61C 1.238G 16GB 300W PCIe2 BOX PeakPerf DP 1.2TFLOPS ;

AOC-IBH-X3QS Mellanox ConnectX3 IB FDR; Single port 4x FDR/QDR IB or 10GbE.

Supermicro Blade 3xGPU K40 (Atlas), blade06

2xCPU IvyBridge-EP E5-2695v2 12C 115W 2.40G 30M 8.00GT/sec;

8x16GB DDR3-1600 ECC REG DIMM 1.5v ;

SATA DOM 32GB MLC Vertical type;

3xGPU NVIDIA TESLA K40 SXM 2880 CUDA Cores PeakPerf DP/SP 1,4/4 TF 12GB 245W PCI-E 3.0;

AOC-IBH-X3QS Mellanox ConnectX3 IB FDR Single port 4x FDR/QDR IB or 10GbE.

Supermicro Blade GPU K20X, Phi 5110P, bade02

2xCPU IvyBridge-EP E5-2695v2 12C 115W 2.40G 30M 8.00GT/sec LGA2011;

8x16GB DDR3-1600 ECC REG DIMM 1.5v;

SATA DOM 32GB MLC Vertical type;

GPU NVIDIA TESLA K20X 2688 CUDA Cores PeakPerf DP/SP 1,31/3,95 TF 6GB 235W PCI-E 2.0;

Phi 5110P 60C 1.053G 8G 225W PCIe2 FHFLDS PeakPerf DP 1.011TFLOPS ;

AOC-IBH-X3QS Mellanox ConnectX3 IB FDR; Single port 4x FDR/QDR IB or 10GbE.

Supermicro Blade 3xGPU K40 (Atlas), blade07

2xCPU IvyBridge-EP E5-2695v2 12C 115W 2.40G 30M 8.00GT/sec;

8x16GB DDR3-1600 ECC REG DIMM 1.5v ;

SATA DOM 32GB MLC Vertical type;

3xGPU NVIDIA TESLA K40 SXM 2880 CUDA Cores PeakPerf DP/SP 1,4/4 TF 12GB 245W PCI-E 3.0;

AOC-IBH-X3QS Mellanox ConnectX3 IB FDR Single port 4x FDR/QDR IB or 10GbE.

Supermicro Blade 2xPhi 7120P, blade03

2xCPU IvyBridge-EP E5-2695v2 12C 115W 2.40G 30M 8.00GT/sec LGA2011;

8x16GB DDR3-1600 ECC REG DIMM 1.5v;

SATA DOM 32GB MLC Vertical type;PCI-E Intel Xeon

2xPhi 7120P 61C 1.238G 16GB 300W PCIe2 BOX PeakPerf DP 1.2TFLOPS ;

AOC-IBH-X3QS Mellanox ConnectX3 IB FDR; Single port 4x FDR/QDR IB or 10GbE.

Supermicro Blade 3xGPU K40 (Atlas), blade05

2xCPU IvyBridge-EP E5-2695v2 12C 115W 2.40G 30M 8.00GT/sec;

8x16GB DDR3-1600 ECC REG DIMM 1.5v ;

SATA DOM 32GB MLC Vertical type;

3xGPU NVIDIA TESLA K40 SXM 2880 CUDA Cores PeakPerf DP/SP 1,4/4 TF 12GB 245W PCI-E 3.0;

AOC-IBH-X3QS Mellanox ConnectX3 IB FDR Single port 4x FDR/QDR IB or 10GbE.

Supermicro Blade 2xGPU K80, blade08

2xCPU IvyBridge-EP E5-2695v3 14C 120W 2.30G 35M 9.60GT/sec LGA2011-3;

8x64GB DDR4-2133 ECC REG DIMM 1.5v;

SATA DOM 32GB MLC Vertical type;

2xGPU NVIDIA TESLA K80 (2x Kepler GK210) 4992 CUDA Cores 24GB GDDR5 480 GB/sec PCIe 3.0 - PeakPerf DP/SP 1.87/5.6 TF;

Mellanox ConnectX-3 VPI InfiniBand Adapter Card, Single-Port QSFP FDR10 IB (40Gb/s) and 10GbE.

Supermicro Blade 3xGPU K40 (Atlas), blade04

2xCPU IvyBridge-EP E5-2695v2 12C 115W 2.40G 30M 8.00GT/sec;

8x16GB DDR3-1600 ECC REG DIMM 1.5v ;

SATA DOM 32GB MLC Vertical type;

3xGPU NVIDIA TESLA K40 SXM 2880 CUDA Cores PeakPerf DP/SP 1,4/4 TF 12GB 245W PCI-E 3.0;

AOC-IBH-X3QS Mellanox ConnectX3 IB FDR Single port 4x FDR/QDR IB or 10GbE.

Supermicro Blade 2xGPU K80, blade09

2xCPU IvyBridge-EP E5-2695v3 14C 120W 2.30G 35M 9.60GT/sec LGA2011-3;

8x64GB DDR4-2133 ECC REG DIMM 1.5v;

SATA DOM 32GB MLC Vertical type;

2xGPU NVIDIA TESLA K80 (2x Kepler GK210) 4992 CUDA Cores 24GB GDDR5 480 GB/sec PCIe 3.0 - PeakPerf DP/SP 1.87/5.6 TF;

Mellanox ConnectX-3 VPI InfiniBand Adapter Card, Single-Port QSFP FDR10 IB (40Gb/s) and 10GbE.

Приложение 2. Исходный код TestEm12 (файл TestEm12.cc)

```
//
// ********************************************************************
// * License and Disclaimer                                           *
// *                                                                  *
// * The  Geant4 software  is  copyright of the Copyright Holders  of *
// * the Geant4 Collaboration.  It is provided  under  the terms  and *
// * conditions of the Geant4 Software License,  included in the file *
// * LICENSE and available at  http://cern.ch/geant4/license .  These *
// * include a list of copyright holders.                             *
// *                                                                  *
// * Neither the authors of this software system, nor their employing *
// * institutes,nor the agencies providing financial support for this *
// * work  make  any representation or  warranty, express or implied, *
// * regarding  this  software system or assume any liability for its *
// * use.  Please see the license in the file  LICENSE  and URL above *
// * for the full disclaimer and the limitation of liability.         *
// *                                                                  *
// * This  code  implementation is the result of  the  scientific and *
// * technical work of the GEANT4 collaboration.                      *
// * By using,  copying,  modifying or  distributing the software (or *
// * any work based  on the software)  you  agree  to acknowledge its *
// * use  in  resulting  scientific  publications,  and indicate your *
// * acceptance of all terms of the Geant4 Software license.          *
// ********************************************************************
//
/// \file electromagnetic/TestEm12/TestEm12.cc
/// \brief Main program of the electromagnetic/TestEm12 example
//
// $Id: TestEm12.cc 85260 2014-10-27 08:53:35Z gcosmo $
//
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo...
...
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo...
...
```

```
#ifdef G4MULTITHREADED
#include "G4MTRunManager.hh"
#else
#include "G4RunManager.hh"
#endif

#include "G4UImanager.hh"
#include "Randomize.hh"

//G4-TBB interfaces
#include "tbbUserWorkerInitialization.hh"
#include "tbbMasterRunManager.hh"
#include "G4Threading.hh"

#include "DetectorConstruction.hh"
#include "PhysicsList.hh"
#include "ActionInitialization.hh"
#include "SteppingVerbose.hh"

//TBB includes
#include <tbb/task_scheduler_init.h>
#include <tbb/task.h>

#ifdef G4VIS_USE
 #include "G4VisExecutive.hh"
#endif

#ifdef G4UI_USE
#include "G4UIExecutive.hh"
#endif

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo...
...

G4ThreadFunReturnType startWork(G4ThreadFunArgType arg)
{
    tbb::task_list* tasks = static_cast<tbb::task_list*>(arg);
```

```
    //We assume at least one /run/beamOn was executed, thus the tasklist is
now filled,
    //lets start TBB
    try {
        std::cout<<"Now spawn work and waiting"<<std::endl;
        tbb::task::spawn_root_and_wait( *tasks );
    } catch(std::exception& e) {
        std::cerr<<"Error occurred. Error test
is:\""<<e.what()<<"\""<<std::endl;
    }
    return static_cast<G4ThreadFunReturnType>(0);
}

//....ooo00000ooo........ooo00000ooo........ooo00000ooo........ooo00000ooo...
...

int main(int argc,char** argv) {

  //choose the Random engine
  G4Random::setTheEngine(new CLHEP::RanecuEngine);

  //Construct the default run manager

  //=== TBB engine initialization
  tbb::task_scheduler_init init( G4Threading::G4GetNumberOfCores() );
  tbb::task_list tasks;

  tbbMasterRunManager* runManager = new tbbMasterRunManager;



  G4VSteppingVerbose::SetInstance(new SteppingVerbose);


  //set mandatory initialization classes
  DetectorConstruction* det = new DetectorConstruction;
  runManager->SetUserInitialization(det);
```

```
  PhysicsList* phys = new PhysicsList;
  runManager->SetUserInitialization(phys);

  runManager->SetUserInitialization(new ActionInitialization(det, phys));

  //get the pointer to the User Interface manager
  G4UImanager* UI = G4UImanager::GetUIpointer();

  if (argc!=1)   // batch mode
    {
     G4String command = "/control/execute ";
     G4String fileName = argv[1];
     UI->ApplyCommand(command+fileName);
    }

  else           //define visualization and UI terminal for interactive mode
    {


#ifdef G4UI_USE
      G4UIExecutive * ui = new G4UIExecutive(argc,argv);
      ui->SessionStart();
      delete ui;
#endif

    }

  G4Thread aThread;
  G4THREADCREATE(&aThread,startWork,static_cast<G4ThreadFunArgType>(&tasks));

  //Wait for work to be finised
  G4THREADJOIN(aThread);
  //job termination
  //
  delete runManager;

  return 0;
}
```

```
//....ooo00000ooo........ooo00000ooo........ooo00000ooo........ooo00000ooo.....
```

Приложение 3. Исходный код Underground Physics (файл DMX.cc)

```
//
// ********************************************************************
// * License and Disclaimer                                           *
// *                                                                  *
// * The  Geant4 software  is  copyright of the Copyright Holders  of *
// * the Geant4 Collaboration.  It is provided  under  the terms  and *
// * conditions of the Geant4 Software License,  included in the file *
// * LICENSE and available at  http://cern.ch/geant4/license .  These *
// * include a list of copyright holders.                             *
// *                                                                  *
// * Neither the authors of this software system, nor their employing *
// * institutes,nor the agencies providing financial support for this *
// * work  make  any representation or  warranty, express or implied, *
// * regarding  this  software system or assume any liability for its *
// * use.  Please see the license in the file  LICENSE  and URL above *
// * for the full disclaimer and the limitation of liability.         *
// *                                                                  *
// * This  code  implementation is the result of  the  scientific and *
// * technical work of the GEANT4 collaboration.                      *
// * By using,  copying,  modifying or  distributing the software (or *
// * any work based  on the software)  you  agree  to acknowledge its *
// * use  in  resulting  scientific  publications,  and indicate your *
// * acceptance of all terms of the Geant4 Software license.          *
// ********************************************************************
//
//
// ------------------------------------------------------------
//   GEANT 4 - Underground Dark Matter Detector Advanced Example
//
//      For information related to this code contact: Alex Howard
//      e-mail: alexander.howard@cern.ch
// ------------------------------------------------------------
// Comments
//
//            Underground Advanced example main program
```

```
//               by A. Howard and H. Araujo
//                    (27th November 2001)
//
// main program
// ------------------------------------------------------------

#ifdef G4MULTITHREADED
#include "G4MTRunManager.hh"
#else
#include "G4RunManager.hh"
#endif

#include "G4UImanager.hh"
#include "Randomize.hh"

#ifdef G4VIS_USE
#include "G4VisExecutive.hh"
#endif

#ifdef G4UI_USE
#include "G4UIExecutive.hh"
#endif

#include "DMXAnalysisManager.hh"
#include "DMXDetectorConstruction.hh"
#include "DMXPhysicsList.hh"
#include "DMXActionInitializer.hh"

int main(int argc,char** argv) {

  // choose the Random engine
  G4Random::setTheEngine(new CLHEP::RanecuEngine);

  // Construct the default run manager
#ifdef G4MULTITHREADED
  G4MTRunManager* runManager = new G4MTRunManager;
  //runManager->SetNumberOfThreads(2);
#else
```

```
  G4RunManager* runManager = new G4RunManager;
#endif

  // set mandatory initialization classes
  runManager->SetUserInitialization(new DMXDetectorConstruction);
  runManager->SetUserInitialization(new DMXPhysicsList);
  runManager->SetUserInitialization(new DMXActionInitializer());

#ifdef G4VIS_USE
  // visualization manager
  G4VisManager* visManager = new G4VisExecutive;
  visManager->Initialize();
#endif


#ifdef DMXENV_GPS_USE
  G4cout << " Using GPS and not DMX gun " << G4endl;
#else
  G4cout << " Using the DMX gun " << G4endl;
#endif


  //Initialize G4 kernel
  runManager->Initialize();

  // get the pointer to the User Interface manager
  G4UImanager* UImanager = G4UImanager::GetUIpointer();

  // Define UI session for interactive mode.
  if(argc == 1)
    {
#ifdef G4UI_USE
      G4UIExecutive* ui = new G4UIExecutive(argc, argv);
#ifdef G4VIS_USE
      UImanager->ApplyCommand("/control/execute initInter.mac");
#endif
      ui->SessionStart();
      delete ui;
```

```
#endif
    }
  // Batch mode
  else
    {
      G4String command = "/control/execute ";
      G4String fileName = argv[1];
      UImanager->ApplyCommand(command+fileName);
    }

  //Close-out analysis:
  // Save histograms
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  man->Write();
  man->CloseFile();
  // Complete clean-up
  delete G4AnalysisManager::Instance();

#ifdef G4VIS_USE
  if(visManager) delete visManager;
#endif
  delete runManager;

  return 0;
}
```